Rui Rodrigo Garcia Alves

Licenciado em Engenharia Informática

Temporal Gisplay

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

Orientador: João Carlos Gomes Moura Pires, Assistant Professor, NOVA University of Lisbon

Co-orientador: Fernando Pedro Reino da Silva Birra, Assistant Professor, NOVA University of Lisbon

Júri

Presidente: Prof. Doutor Pedro Medeiros
Arguente: Prof. Doutor Daniel Gonçalves
Vogal: Prof. Doutor João Moura Pires

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Dezembro, 2017

Temporal Gisplay

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "unlthesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/unlthesis [2] http://www.di.fct.unl.pt

Agradecimentos

À Universidade Nova de Lisboa e à NOVA LINCS. Aos meus pais pelo apoio demonstrado, aos orientadores pela orientação e conhecimento partilhado e aos meus colegas pelo *feedback* construtivo.

Resumo

Nos dias que correm devido à ubiquidade dos dispositivos que recolhem informação, desde computadores a telemóveis, são produzidos uma grande quantidade de dados, sendo que estes dados têm a componente temporal e também espacial porque os dispositivos são móveis e têm eles próprios capacidade de recolha de localização.

Os mapas temáticos são uma importante ferramenta para análise de fenómenos ao longo do tempo e espaço e os mesmos são ainda um problema por resolver.

Verificou-se a inexistência de uma API para construção de mapas temáticos na web, que permita interação, lidando com a componente temporal, assente nos melhores princípios da cartografia e visualização de dados e que tenha um desempenho que permita lidar com milhões de pontos assegurando uma interação fluída.

Das APIs avaliadas durante esta dissertação todas elas tinham um ou mais aspetos que não satisfaziam alguma das caraterísticas acima referidas, sendo de mencionar particularmente a API Gisplay pois a mesma adota a tecnologia WebGL como solução para obter desempenho requerido. Apesar disto, a API Gisplay não suporta a componente temporal e tem fortes limitações sob o ponto de vista de visualização de variáveis visuais.

Nesta dissertação foi proposta e implementada a API Temporal Gisplay, uma API JavaScript, com alto nível de abstração e facilmente extensível com as caraterísticas acima referidas. A avaliação realizada demonstra que a API Temporal Gisplay permite lidar com milhões de pontos, possibilita o uso de várias variáveis visuais, permite lidar com a componente temporal e possui um conjunto de mapas temáticos que permitem visualizar cenários bastante distintos.

Palavras-chave: Mapas Temáticos, WebGL, Visualização Analítica, Tempo

Abstract

Nowadays, due to the ubiquity of the devices that gather information, ranging from computers to mobile phones, an increasing amount of data is produced, and such data contains the temporal and spatial component because those devices are mobile and have their own capability of collecting location information.

Thematic maps are one important tool for the analysis of phenomena through time and space and they're a problem remaining to be solved.

It was clear that there wasn't an API to build thematic maps based on the web, that enables interaction, dealing with millions of points, using the best principles of cartography and data visualization while allowing to deal with millions of points and still maintining fluid interaction.

All the evaluated APIs during this dissertation had one or more aspects that wouldn't satisfy some of the above characteristics, with a particular mention to the API Gisplay because this API adopts the WebGL technology as a solution to obtain the required performance. Despite this, the Gisplay API does not support the temporal component and has huge limitations from the point of view of visual variables visualization.

On this dissertation was proposed and implemented the API Temporal Gisplay, an JavaScript API, with a high level of abstraction and easily extensible with the uppermention characteristics. The evaluation that was done shows that the API Temporal Gisplay is able to deal with millions of points, enables the use of multiple visual variables, is able to use the temporal component and has a set of thematic maps that can be used to visualize different scenarios.

Keywords: Thematic Maps, WebGL, Analytical Visualization, Time

Índice

Capítulo 1

Introdução

1.1 Contexto e Motivação

Com o aumento que se verifica da quantidade de dados espácio-temporais que são recolhidos, é necessário ter formas de tratar e visualizar estes dados. Exemplos de eventos espácio-temporais são a evolução de casos epidemiológicos, crimes ocorridos num país, incêndios que deflagraram num distrito, entre outros. Por exemplo, as centenas de incêndios que deflagraram no dia 15 de Outubro de 2017 em Portugal Continental[1], são eventos espácio-temporais e a sua análise é difícil pelas várias componentes que os caraterizam.

Para visualizar os dados espácio-temporais são muitas vezes usados os mapas temáticos, sendo que este tipo de mapas é utilizado não só para comunicação, mas também em contextos analíticos.

O problema de criar mapas temáticos usando tecnologias que permitam a sua fácil interação no browser ainda não está resolvido. Existem APIs Javascript como Leaflet, Google Maps ou Open Layers que permitem simplificar o processo de visualização de mapas interativos nos browsers. Estas APIs têm vindo a ser usadas para criar mapas temáticos para variadíssimas tarefas analíticas. Contudo, estas têm uma performance fraca quando o número de dados a ser mostrados é considerável, pois estas usam objetos DOM para a visualização e sendo assim não permite escalabilidade.

Numa dissertação de mestrado anterior foi proposta e implementada a API Gisplay [17] que tem como foco principal oferecer a construção de mapas temáticos na web de forma rápida e fácil, apostando num alto nível de abstração, pretendendo também ter capacidade de lidar com grandes quantidades de dados enquanto que ao mesmo tempo garante que a solução mantém a interatividade desejada. Esta API é uma API Javascript de alto nível que se dedica à construção de mapas temáticos utilizando tecnologia WebGL.

[1] https://pt.wikipedia.org/wiki/Incêndios_florestais_em_Portugal_de_outubro_de_2017

A API suporta operações interativas como por exemplo pan, zoom e filtragem em mapas temáticos somente com a componente espacial.

A maior parte das soluções de mapas temáticos que estão atualmente disponíveis, não incorporam a componente temporal. A própria API Gisplay carece dessa funcionalidade, tendo apenas um tipo de mapas que pode ser usado para mostrar a evolução temporal dos dados. É então fundamental criar uma API que tenha capacidade de criar mapas temáticos com componente temporal associada num contexto web.

1.2 Objetivos

O principal objetivo desta dissertação foi a adição da componente temporal em mapas temáticos. Esta componente tem grande impacto na forma como os dados se vão organizar, sendo que os mesmos podem ou não ter entidades. Neste contexto, existência de entidade significa que existem múltiplos registos associados a um individuo ou coisa, sendo que, tipicamente cada registo está associado a um momento temporal dessa mesma entidade. A componente temporal pode ainda ter um arranjo linear ou cíclico com implicações importantes em termos da visualização e da navegação nesta nova dimensão. Outra necessidade que a componente temporal acarreta é a de existir uma forma de a controlar e de navegar na mesma, sendo para tal usado um controlador temporal que permita navegar em instantes ou intervalos.

Com o aumento da quantidade de dados, da sua complexidade e da forma de processamento terá de ser implementado um parser CSV para conseguir obter uma performance que permita lidar com milhões de pontos.

Em termos de mapas temáticos, a API terá de oferecer uma variedade satisfatória que permita lidar com várias primitivas diferentes. Para esses mapas serão fornecidos os melhores valores por omissão em termos de cor e cálculo de classes. Os mapas temáticos possuirão várias projeções permitindo visualizar várias zonas geográficas em simultâneo. As opções fornecidas terão de permitir um alto nível de personalização dos mapas temáticos e das variáveis visuais.

Os mapas temáticos não se fazem sem a existência de variáveis visuais, sendo que a API permitirá o uso da cor, tamanho, forma, textura e orientação. Estas variáveis visuais poderão ainda ser usadas em simultâneo no mesmo mapa temático, existindo as respetivas legendas. Estas legendas serão de fácil utilização sendo efetivas do ponto de vista visual e analítico e fornecerão mecanismos para seleção e filtragem dos dados visualizados.

Tudo isto, resultará em que a API seja facilmente extensível e seja capaz de lidar com milhões de pontos mas mesmo assim terá de ser capaz de manter a interatividade necessária para permitir a análise através das operações de pan, zoom e seleção.

1.3 Contribuições

O foco desta dissertação foi a introdução da componente temporal numa API de mapas temáticos. Para a introdução da componente temporal foi necessário perceber quais os mapas que contém a componente temporal. Foi possível perceber que já existe uma vasta literatura sobre este tema quando se fez uma pesquisa e análise dos principais mapas pertencentes a esta categoria.

Na sequência do uso da componente temporal foi introduzido um parser para ficheiros CSV que permite lidar com milhões de pontos permitindo obter a performance que pretendíamos para a API. Este parser tem uma performance bastante superior aos existentes atualmente, sendo desta forma uma contribuição de relevo que resulta desta dissertação.

Foram formuladas as tarefas conceptuais para a construção de um mapa temático, tendo em conta o uso da componente temporal.

Em relação às APIs para mapas temáticos na Web, foi feita uma pesquisa e análise crítica de várias APIs que podem ser utilizadas este fim. Nas APIs encontradas, o principal problema identificado, foi o fraco desempenho com grandes quantidades de dados e também a inexistência de uma API *client-side* focada na criação de mapas temáticos que permitisse o uso da componente temporal.

Foi implementada a API Temporal Gisplay tendo em conta os objetivos prontamente propostos. Por fim, é de referir que foi feita a avaliação da API Temporal Gisplay para aferir se os objetivos propostos foram cumpridos.

Nas contribuições é ainda de referir o artigo[18] que foi produzido no contexto da conferência ICCSA 2017.

1.4 Estrutura do Documento

Este documento tem a seguinte estrutura:

- **Capítulo 1 - Introdução**
  Oferece uma visão global do documento, dando um contexto e motivação para a realização desta dissertação, os objetivos a cumprir durante a mesma e quais foram as principais contribuições feitas.

- **Capítulo 2 - Estado da Arte**
  Apresenta o estado da arte, começando por introduzir conceitos importantes da cartografia e da visualização em mapas temáticos, tais como projeções, legendas, variáveis visuais e esquemas de cores. Apresenta depois os vários tipos de mapas temáticos existentes. De seguida temos o tempo em mapas temáticos, onde são apresentadas as formas de adicionar o tempo nos mapas temáticos bem como uma lista extensa de mapas temáticos que lidam com a componente temporal. Por fim, temos uma secção sobre os mapas temáticos na Web em que são vistas várias APIs que permitem criar mapas temáticos bem como quais são as suas limitações.

- **Capítulo 3 - Objetivos**
  Neste capítulo são explicados em detalhe quais os principais objetivos que esta dissertação propõe cumprir.

- **Capítulo 4 - Abordagem**
  Apresenta-se a abordagem que foi adotada. Nomeadamente, começamos por redefinir as tarefas conceptuais. Depois apresenta-se a arquitetura da API Temporal Gisplay, bem como as APIs de alto nível e de nível intermédio.

- **Capítulo 5 - Implementação**
  Este capítulo visa mostrar todos os aspetos de implementação tidos em conta, desde tratamento dos dados, organização dos dados após processamento, cálculo de classes, mapas de fundo, múltiplas projeções, mapas temáticos implementados, interação temporal, interação espacial e legenda.

- **Capítulo 6 - Avaliação**
  Divulgados os dados relativos à avaliação da API Temporal Gisplay, nomeadamente como é que esta se compara com outras existentes e se cumpre os objetivos que foram propostos.

- **Capítulo 7 - Conclusões e Trabalho Futuro**
  Contém as conclusões obtidas para esta dissertação e apresenta possíveis direções para realização em trabalho futuro.

Capítulo 2

Estado da Arte

Neste capítulo primeiro teremos uma pequena introdução à cartografia e mapas. Depois serão apresentadas as projeções, sendo este um conceito fundamental para a cartografia. De seguida é discutido o papel da legenda na leitura e compreensão de um mapa. Por fim é apresentada uma pequena introdução aos tipos de mapas existentes, dos quais se destacam para a realização desta dissertação, os mapas temáticos.

Na secção 2.2 irá ser feita uma pequena introdução aos mapas temáticos, depois são dados a conhecer alguns dos conceitos subjacentes aos mapas temáticos que são muito importantes para esta dissertação, sendo estes conceitos: os esquemas de cores, variáveis visuais e os tipos de dados usados nas representações feitas em mapas temáticos. De seguida, está presente o estado da arte em relativo a mapas temáticos. Depois discute-se a inclusão do tempo, sendo apresentadas várias técnicas que retratam a componente temporal, agrupando-as em técnicas que trabalham a duas dimensões e técnicas que trabalham a três dimensões.

Por fim, temos uma secção sobre mapas temáticos na web, sendo apresentadas as formas mais usuais para construir os mapas temáticos bem como qual a abordagem levada a cabo pela API Gisplay.

2.1 Cartografia

A cartografia é a ciência responsável pela elaboração, produção, divulgação e estudo de mapas [64]. Os mapas têm como objetivo transmitir informação espacial através da representação gráfica de uma região ou superfície terrestre, sendo usados principalmente para (1) localizar sítios, (2) medir distâncias, (3) planear viagens e (4) visualizar dados georreferenciados.

O desenho de um mapa é severamente influenciado pelos dados a visualizar, sendo

peça chave na escolha da legenda e escala do mapa. Os dados a visualizar num mapa referem-se a objetos ou fenómenos no espaço, sendo representados através de pontos, linhas ou áreas.

Como os mapas são usados para produzir uma representação gráfica total ou parcial da superfície terrestre, é fundamental perceber como podemos projetar a superfície terrestre num mapa.

2.1.1 Projeções

Como sabemos, a terra tem uma forma aproximadamente esférica, então precisamos de representar a superfície esférica da Terra num plano, ou seja, num mapa.

Os mapas que retratam a superfície da Terra usam a projeção para transformar/-mapear as três dimensões da superfície real nas duas dimensões do mapa (superfície plana). As equações matemáticas que são usadas para projetar latitude e longitude em coordenadas no plano são chamadas projeções de mapa.

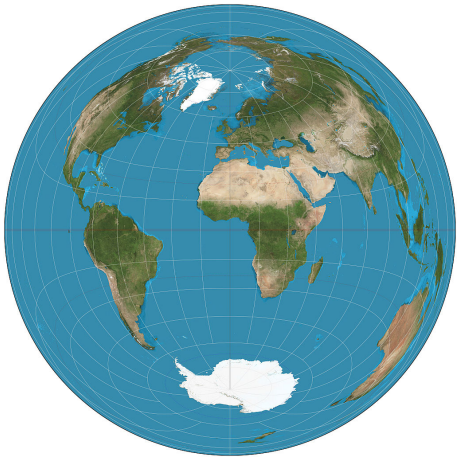
Todas as projeções de mapas distorcem a superfície de alguma maneira. Dependendo do objetivo do mapa, algumas destas distorções são aceitáveis, enquanto que outras não são aceitáveis, logo, diferentes tipos de projeções têm de existir para fazer face a este problema. Cada uma das projeções preservam algumas das propriedades em detrimento de outras [34].

As projeções cartográficas podem ser divididas em três categorias principais, dependendo da figura geométrica usada para a sua construção:

1. Projeção azimutal ou plana - Neste tipo de projeção o mapa é representado a partir de um ponto central de forma radial. É muito utilizado para representar regiões polares, estudos sísmicos e trabalhos relacionados com ondas de rádio. Na figura 2.1a é possível ver a projeção azimutal de Lambert, a qual preserva a área.

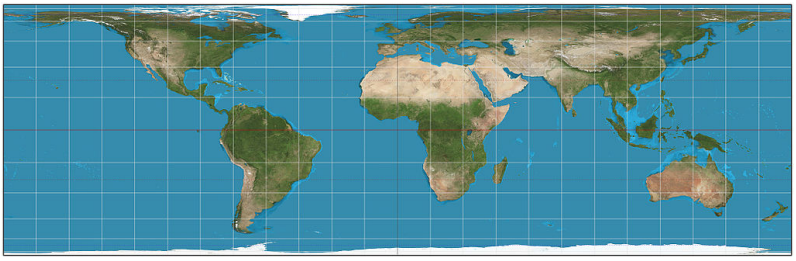
2. Projeção cónica - Mapeia a superfície da esfera na superfície de um cone, que depois é desenrolado para uma superfície plana. É muito utilizada para representar regiões de latitudes médias. Na figura 2.1b é possível ver um exemplo de uma projeção cónica de Lambert.

3. Projeção cilíndrica - Representam a Terra a partir de um cilindro estendido. Os meridianos são mapeados para linhas verticais com espaçamento igual e os círculos de latitude (paralelos) são mapeados para linhas horizontais. Na figura 2.1c, é possível ver um exemplo de uma projeção cilíndrica, a qual preserva a área.



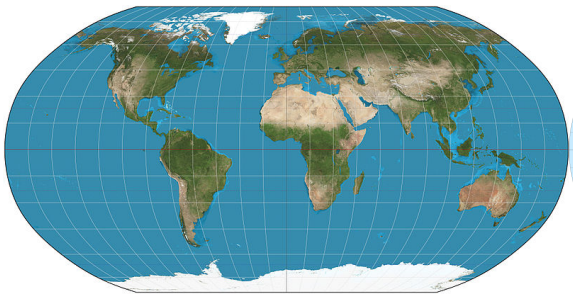
(a) Projeção Azimutal de Lambert [60]

(b) Projeção Cónica de Lambert [58]

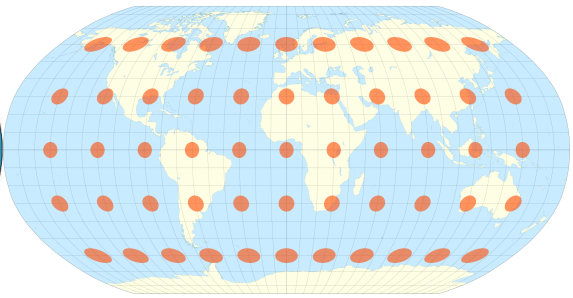


(c) Projeção Cilíndrica de Lambert [57]

Figura 2.1: Exemplos de Projeções



(a) Projeção de Robinson [61]

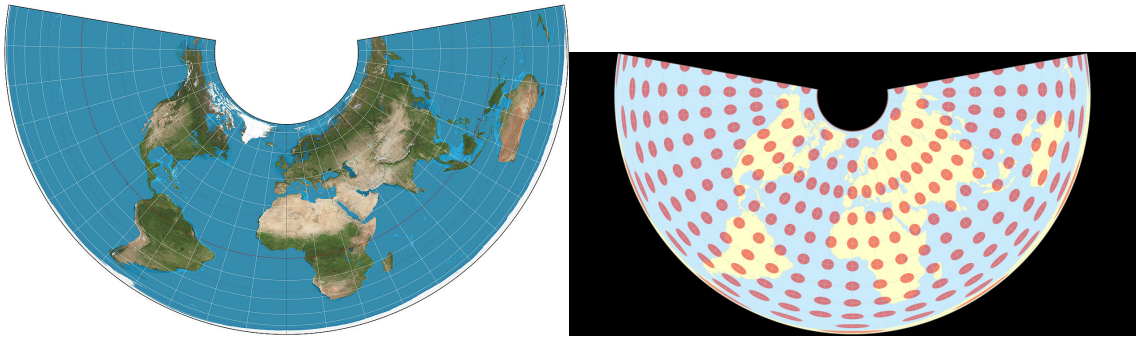


(b) Deformações da geometria na projeção de Robinson [24]

Figura 2.2: Projeção de Robinson (a) e respetivas deformações (b)

A projeção de Robinson, que é possível ver na figura 2.2a, é a única projeção que foi construída usando um método único e díspar de todas as outras projeções existentes, pelo que não se pode colocar perfeitamente em nenhuma das categorias acima descritas. A projeção de Robinson não foi desenvolvida a partir de novas fórmulas geométricas para converter a latitude e longitude da superfície do modelo terrestre para uma mapa. Em vez disso, *Arthur H. Robinson* usou em grande parte simulações computacionais e fez tentativa e erro de modo a obter uma projeção que permite mostrar todo o globo como uma imagem plana de uma forma natural. Na figura 2.2b, é possível ver as deformações

provocadas pela projeção de Robinson, sendo possível verificar que tal como qualquer outra projeção possui deformação da geometria, no entanto mantém a distorção a um nível baixo por todo o mapa.



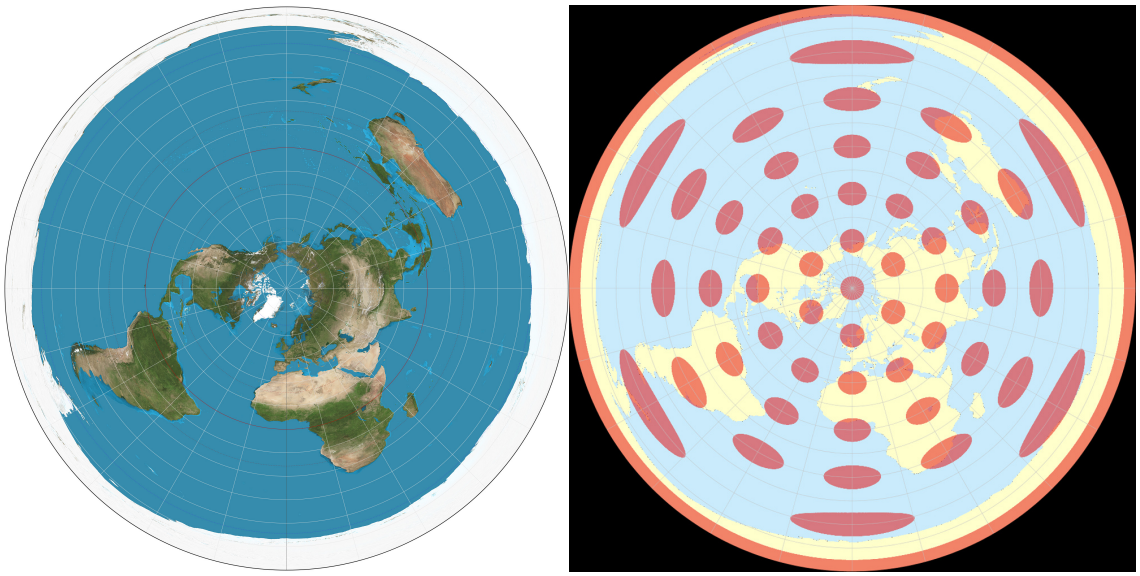
(a) Projeção de Albers [59]

(b) Deformações da geometria na projeção de Albers [33]

Figura 2.3: Projeção de Albers (a) e respetivas deformações (b)

Na figura 2.3a, pode ser vista a projeção de Albers, que é um exemplo de uma projeção cónica. Na figura 2.3b estão dispostas as deformações que esta mesma projeção possui.

Depois, na figura 2.4a a projeção azimutal equidistante, a qual tem a propriedade importante de todos os pontos no mapa estarem a uma distancia proporcional ao ponto central.



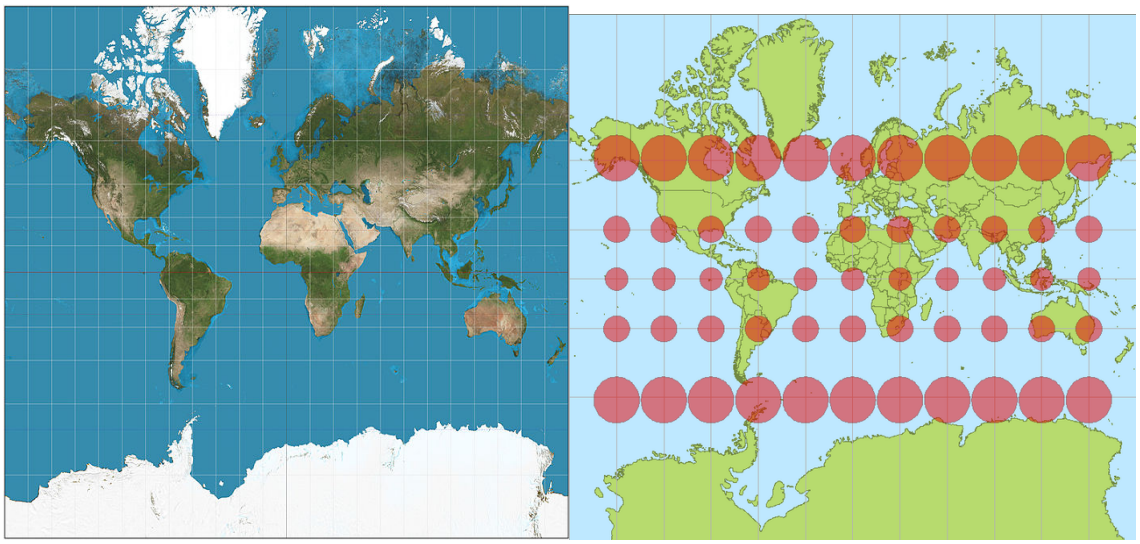
(a) Projeção Azimutal Equidistante [56]

(b) Deformações da geometria na projeção Azimutal Equidistante [32]

Figura 2.4: Projeção Azimutal Equidistante (a) e respetivas deformações (b)

Por fim, temos um exemplo muito conhecido de projeção cilíndrica, a Projeção de Mercator, que é possível ver na figura 2.5a. Esta projeção foi originalmente criada para ajudar na navegação marítima, e tem a limitação de não conseguir representar os pólos

[34, p.77]. A projeção de Mercator é a mais importante para esta dissertação, pois como veremos todos os provedores de mapas de fundo que foram adicionados à API Gisplay usam a projeção Web Mercator, que é uma variante da projeção de Mercator.



(a) Projeção de Mercator [19]

(b) Deformações da geometria na projeção de Mercator [36]

Figura 2.5: Projeção de Mercator (a) e respetivas deformações (b)

Na figura 2.5b é possível visualizar as deformações que ocorrem na projeção de Mercator. Neste caso, localizações próximas do equador têm pequenas distorções enquanto que zonas mais próximas dos pólos sofrem deformações bastante significativas.

Depois de termos visto várias possibilidades de projetar o mundo real num mapa, é importante saber como ler um mapa de modo a perceber a informação que este pretende transmitir.

2.1.2 Legenda

Para ler um mapa precisamos de entender, primeiramente, o que é a legenda do mapa, bem como a escala na qual o mapa se encontra desenhado.

A legenda de um mapa explica os símbolos e cores que estão presentes no referido mapa. Por outras palavras, a legenda de um mapa é o componente que transmite toda a informação necessária para que o mapa faça sentido. Na figura 2.6 está disposto um exemplo da legenda de um mapa. Nesta legenda estão identificados quais os símbolos e cores usados para estradas, cidades, lagos, entre outros.

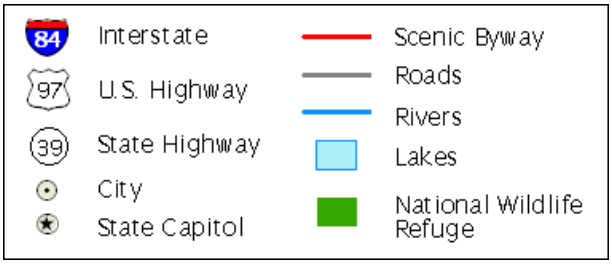


Figura 2.6: Exemplo de legenda de um mapa. Retirado de [40].

A escala num mapa mostra a relação entre as distâncias no mapa e as distâncias na superfície terrestre. A escala pode ser mostrada graficamente, através de um gráfico de barras graduado por distâncias. Este método é útil porque mantém-se preciso, mesmo que o mapa seja aumentado ou reduzido. Uma segunda alternativa para mostrar a escala de um mapa é o uso verbal, que simplesmente usa palavras para descrever o rácio entre a escala do mapa e o mundo real(e.g. 3 centímetros = 10 metros). A última alternativa para mostrar a escala de um mapa é o método fracionário. Neste método usa-se uma fração para descrever o rácio entre o mapa e o mundo real(e.g. 1:5000) sendo que neste exemplo uma unidade de distância no mapa equivale a cinco mil unidades de distância no mundo real. Na figura 2.7 é possível ver um exemplo de cada um dos tipos de escalas: verbal, fração e gráfico de barras.

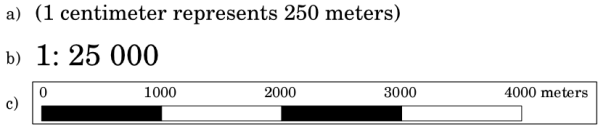


Figura 2.7: Os três tipos de escala: verbal, fração e barra, respetivamente. Retirado de [41]

2.1.3 Tipos de Mapas

Existem diversos tipos de mapas, sendo que importa identificar os mais importantes, seja pelo seu amplo uso em várias áreas ou pela sua relevância para o conhecimento e história humana [66].

Em primeiro lugar temos os mapas físicos, que representam caraterísticas físicas do território, como por exemplo linhas de água, relevo, altitude, clima e vegetação. Este tipo de mapas é usado maioritariamente pelas ciências da Terra para registar graficamente características físicas de um local.

Os mapas políticos representam divisões territoriais com fins administrativos ou legais nos quais os limites são de grande importância enquanto que outras caraterísticas como relevo, linhas de água e afins, que eram muito importantes nos mapas físicos, são agora de importância secundária. Os mapas políticos são usados para mostrar a divisão do território em países, distritos.

Os mapas históricos apresentam as mudanças ocorridas numa determinada região ao longo de um determinado período de tempo. Estes tipos de mapas são muito importantes

para entender alguns acontecimentos históricos, sejam conquistas marítimas ou batalhas terrestres e também são relevantes no contexto desta dissertação porque incorporam a componente temporal.

Os mapas de estradas são mapas usados para mostrar estradas de maior e menor importância (dependendo do tipo de detalhe que o mapa possui), bem como sítios como localização de cidades, aeroportos e outros pontos de interesse como parques ou monumentos. Este é um dos tipos de mapas mais usados porque os humanos foram criando rotas terrestres para se movimentarem de forma mais fácil um pouco por toda a superfície terrestre, sendo estas rotas, ou parte delas, o alvo de representação destes mapas.

Por fim, temos os mapas temáticos que são mapas que se focam num determinado tópico ou tema e que serão alvo de escrutínio profundo na próxima secção deste capítulo.

2.2 Mapas Temáticos

Nesta secção vão ser vistos os mapas temáticos em mais detalhe, sendo estes um ponto chave desta dissertação, bem como quais os tipos de mapas temáticos que existem e como podemos adicionar a componente temporal nestes tipo de mapas.

2.2.1 Introdução

Os mapas temáticos, também conhecidos como mapas de propósito especial, são mapas que ilustram a distribuição geográfica de um tópico ou tema particular [42]. Quando o objetivo central é reunir somente alguns dados ou caraterísticas sobre algo em específico, tal como cultura, vegetação, clima, temperatura, população, usam-se então os mapas temáticos.

Os mapas temáticos são diferentes dos outros tipos de mapas porque estes não mostram só os recursos naturais como rios, cidades, estradas ou divisões políticas, mas por outro lado se estes itens estiverem presentes no mapa temático, então serão considerados informação de fundo e são usados como pontos de referência para realçar o tema a ser disposto no mapa [15].

Os mapas temáticos são usados para três fins principais:

1. Fornecer informações gerais sobre padrões espaciais. Isto é, por exemplo, permitir ao utilizador discernir entre várias áreas geográficas quais são as diferenças entre os valores do atributo que está a ser mapeado nas referidas áreas geográficas.

2. Fornecer informações sobre localizações específicas. Neste caso permitem, por exemplo, que o utilizador verifique para determinada zona geográfica quais são os valores do atributo.

3. Comparar padrões em dois ou mais mapas. Os mapas temáticos permitem que se, por exemplo, tivermos dois mapas diferentes, cada um deles para um momento temporal diferente ou a representar um atributo diferente, o utilizador consiga

diferenciar os padrões que ocorrem entre os dois momentos temporais ou atributos que estão a ser mostrados em cada um dos mapas.

2.2.2 Tipos de Dados

Primeiramente, precisamos de ter maneira de expressar a quantidade ou a qualidade dos dados, para tal usamos as escalas de medida. As escalas de medida estão divididas em: nominal, ordinal, intervalar e razão [46].

A escala nominal é meramente classificativa, permitindo descrever as variáveis sem recurso à quantificação. Esta escala baseia-se no agrupamento e classificação de elementos para a formação de conjuntos distintos. Toda e qualquer fração dos dados que existirem vão-se encaixar numa única categoria. Como exemplo temos as divisões de género em masculino e feminino.

A escala ordinal mantém as caraterísticas da escala nominal, mas tem a capacidade de ordenar os dados (e.g., Não Gosta-0, Gosta-1, Gosta Muito-2).

A escala intervalar é uma forma quantitativa de registar um fenómeno, medindo-o em termos da sua intensidade específica, ou seja, posicionando-o em relação a um valor conhecido arbitrariamente denominado como ponto zero. Tal aferição é realizada definindo-se a unidade de medida a ser usada nessa comparação a partir da diferença entre o valor no ponto zero e um segundo valor conhecido, sendo assim possível conhecer as distâncias entre quaisquer duas posições(números) desta escala. Exemplos clássicos são as escalas de temperatura onde não se pode assumir um ponto zero como ausência de temperatura, ou dizer que uma determinada temperatura é o dobro de outra temperatura.

A escala proporcional ou de razão tem todas as caraterísticas das escalas anteriormente discutidas e ainda fornece um zero absoluto. Esta escala é a mais completa e sofisticada de todas as escalas. Exemplos do uso desta escala são o peso, altura ou medição de distâncias.

Resumindo, a mais simples e limitada das escalas é a escala nominal, permitindo apenas a identificação de categorias. Em seguida, temos a escala ordinal, que permite diferenciar patamares. Com maior alcance temos a escala intervalar, que permite o posicionamento de valores em relação a um ponto arbitrário. Finalmente, a mais poderosa de todas as escalas é a escala de razão, que permite a comparação de valores em termos absolutos.

Na sua forma mais simples, cada observação ou variável de um registo de dados representa um único pedaço de informação. Podemos classificar esta informação como sendo ordinal (numérica) ou nominal (não-numérica) [74, p.52].

Os dados ordinais assumem valores numéricos e podem ser:

1. binários - assumem valores 0 ou 1;

2. discretos - assumem só valores inteiros ou um sub-conjunto de inteiros (e.g.,(2,4,6));

3. contínuos - representam valores reais (e.g., no intervalo[0,5]).

Os dados nominais assumem valores não-numéricos e podem ser:

1. categóricos - um valor num conjunto finito de possibilidades (e.g., azul, verde, vermelho);

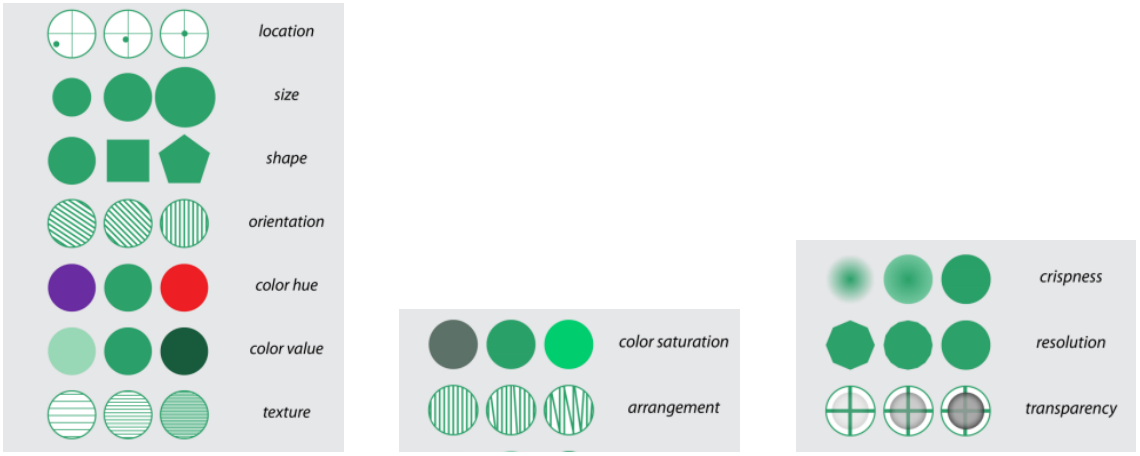
2. classificados - uma variável categórica que tem uma ordem implícita (e.g., pequeno, médio, grande);

3. arbitrários - uma variável que varia num intervalo potencialmente infinito de valores sem qualquer ordenação implícita (e.g., endereços).

2.2.3 Variáveis Visuais

O cartógrafo e professor francês *Jacques Bertin* foi o primeiro a propor um conjunto de variáveis visuais, que são a base da representação gráfica. As variáveis visuais são um conjunto específico de símbolos que podem ser aplicados a dados de maneira a permitirem codificar informação [73].

Como pode ser visto na figura 2.8a, originalmente *Jacques Bertin* propôs sete variáveis visuais que podem ser manipuladas para codificar informação.



(a) Bertin (b) Morrison (c) MacEachren

Figura 2.8: Variáveis Visuais. Retirado de [51]

1. Posição - Descreve a posição de um símbolo relativamente a uma coordenada. A posição é considerada uma variável visual indispensável e toma prioridade em relação às outras. Na cartografia a posição geralmente significa a posição do símbolo em relação ao sistema de coordenadas espacial da projeção utilizada, sendo assim, a posição é usada para representar a componente espacial da informação no processo cartográfico.

2. Tamanho - Descreve a quantidade de espaço ocupado por um símbolo. O tamanho é a primeira variável visual a ser manipulada em mapas de símbolos proporcionais.

3. Forma - Descreve o contorno do símbolo. Esta variável visual é fundamental para o desenho de mapas que façam uso da forma como meio principal para transmitir informação.

4. Orientação - Descreve a rotação do símbolo a partir do "normal" do mapa.

5. Tonalidade da cor(Hue) - Descreve o comprimento de onda dominante do símbolo, no espectro eletromagnético(e.g. Azul, Vermelho, Verde). Esta variável visual é especialmente importante para mapas do tipo Choropleth bem como para qualquer tipo de mapas temáticos que usem a cor para diferenciar categorias ou classes.

6. Valor da cor - Descreve a quantidade de energia emitida ou refletida pelo símbolo. Mais uma vez o valor da cor é importante para os mapas do tipo Choropleth para descrever dados numéricos.

7. Textura - Descreve a grossura do padrão de preenchimento dentro do símbolo.

A lista original de *Bertin* foi depois melhorada e expandida por vários outros autores, como por exemplo Morrison(Saturação da cor e Arranjo), figura 2.8b e MacEachren (Crispness, Resolução e Transparência), figura 2.8c [51].

8. Saturação da cor - Descreve o crescimento no espectro visível do símbolo e é a terceira variável visual associada à cor, além da tonalidade e valor. Os símbolos mais saturados tendem a figurar mais que os menos saturados.

9. Arranjo - Descreve o layout dos elementos gráficos que constituem o conjunto de símbolos do mapa. Esta variável visual difere da textura porque as texturas estão organizadas regularmente independentemente da direção inicial, tamanho ou densidade da textura.

10. Crispness - Descreve a nitidez do limite do símbolo do mapa.

11. Resolução - Descreve a precisão espacial na qual o símbolo do mapa é exibido.

12. Transparência - Descreve a mistura entre um símbolo do mapa e o mapa de fundo ou outros símbolos subjacentes.

As variáveis visuais são muito importantes para permitir passar mais facilmente a informação ao utilizador, sendo uma peça fundamental a ser usada na construção dos mapas temáticos.

2.2.4 Esquemas de Cores

Quando se está a produzir um mapa, é importante que a informação a transmitir seja disposta de forma a que torne a sua compreensão relativamente fácil por parte do utilizador. Para tal é muito importante, que os atributos que forem mapeados usando a cor,

como forma principal de transmissão de informação, estejam dispostos de forma a que o utilizador tenha relativa facilidade em compreender a informação que lhe está a ser transmitida pelo mapa.

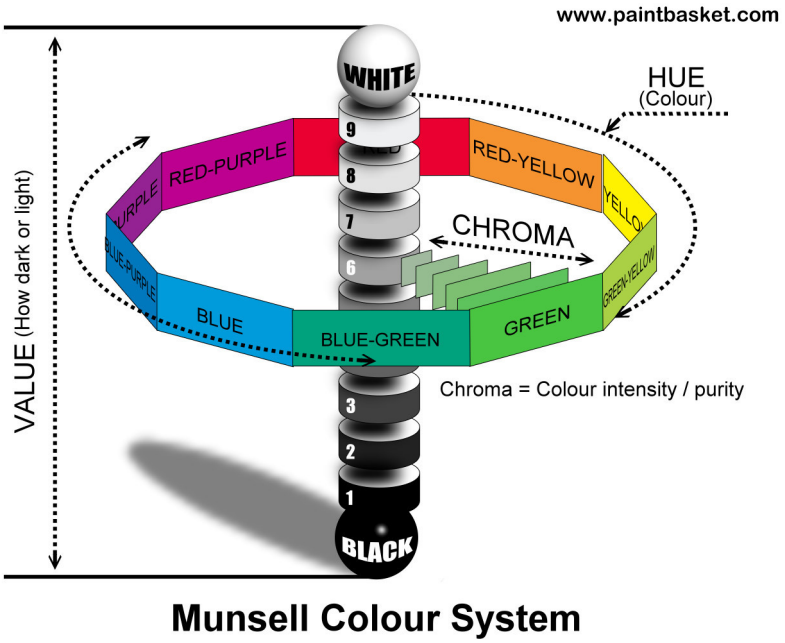


Figura 2.9: Modelo de cores de Munsell. Retirado de [47]

Para que a informação transmitida através da cor num mapa seja de fácil compreensão é necessário que quem tenha criado o mapa entenda bem a teoria da cor. Cor que tenha sido usada corretamente na criação do mapa temático, vai aprimorar e clarificar a apresentação, enquanto que cor que seja usada de forma ineficiente vai atrapalhar e confundir a apresentação [55].

Os nossos olhos através das células da retina conseguem detetar luz vermelha, verde e azul, mas nós não pensamos neste sistema(RGB). Em vez disso, nós pensamos na cor em termos de luminosidade (branco a preto), matiz(vermelho, laranja e amarelo, verde, azul, roxo) e saturação(escuro a brilhante). As variáveis, matiz, luminosidade e saturação, originalmente definidas por *Albert H. Munsell*, são a fundação de qualquer sistema de cores baseados na perceção humana(capacidade humana para reconhecer, organizar e interpretar informação sensorial). No modelo de Munsell, a **matiz** é a qualidade pela qual se distingue uma cor da outra, **luminosidade** é a qualidade pela qual se distingue uma cor mais brilhante de uma mais escura e a **saturação(chroma)** é a qualidade pela qual se consegue distinguir uma tonalidade clara de uma tonalidade escura [54].

Na figura 2.9 é possível ver o modelo de cores de Munsell e é possível ver a matiz(hue), saturação(chroma), luminosidade(value) e para cada uma destas variáveis a forma como variam.

A API Colorbrewer, proposta e implementada por Cynthia Brewer e Mark Harrower em [30] surgiu como forma de abstrair a preocupação da utilização de esquemas de cores

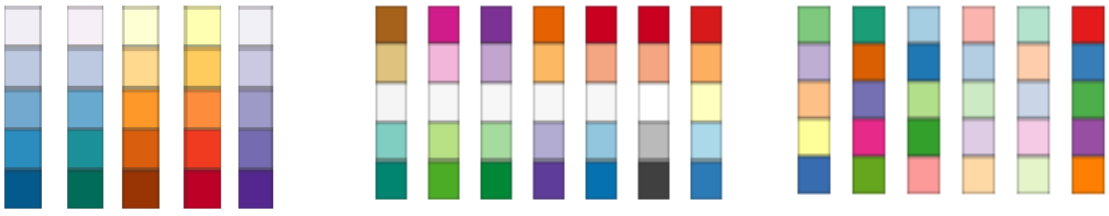
para a construção de mapas temáticos. Esta API deu ainda origem ao site ColorBrewer[1] no qual é possível de uma forma rápida e eficaz, obter esquemas de cores de acordo com vários critérios.

No ColorBrewer são identificados trinta e cinco esquemas de cores que foram desenhados para mapas com três a doze classes. São ainda identificados três tipos de esquemas de cores que devem ser usados consoante as propriedades dos atributos que pretendemos mapear através da cor:

- **Esquema de cores sequenciais**
  Esquemas de cores sequenciais implicam ordem e são adequados para representar dados que variam de um valor baixo para um valor alto numa escala ordinal(e.g., frio, morno, quente) ou numa escala numérica (e.g., classes de idades 0-9, 10-19,etc.). Por norma existe uma variação de luminosidade que dominam estes esquemas, nos quais as cores mais claras representam valores mais baixos e as cores mais escuras representam valores mais altos. Na figura 2.10a é possível ver cinco esquemas de cores sequenciais do conjunto que o ColorBrewer disponibiliza.

- **Esquemas de cores divergentes**
  Os esquemas de cores divergentes devem ser usados quando existe um ponto crítico ou ponto de quebra ao qual se quer dar ênfase. Esta quebra ou classe no meio da sequência deve ser realçada com uma variação de matiz e luminosidade. O ponto de quebra é geralmente obtido nos dados através da média, mediana ou o valor zero. Na figura 2.10b é possível ver sete esquemas de cores divergentes.

- **Esquemas de cores qualitativas**
  Os esquemas de cores qualitativas baseiam-se na diferença de matiz para criar um esquema de cor que não tem ordem implícita, meramente existe diferença de tipo. Os esquemas qualitativos são mais efetivos quando não existe nenhuma variação da saturação ou luminosidade entre as cores escolhidas ou quando esta variação é pequena. Na figura 2.10c é possível ver seis esquemas de cores qualitativas.



(a) Sequencial (b) Divergente (c) Qualitativo

Figura 2.10: Exemplo de esquemas de cores: (a) sequencial, (b) divergente, (c) qualitativo. Retirado de [30].

[1] http://colorbrewer2.org/

2.2.5 Tipos de Mapas Temáticos

Existem vários tipos de mapas temáticos, sendo que estes podem ser classificados consoante o número de variáveis visuais que utilizam para a representação dos dados:

1. **univariate**(uma variável visual), se todos os dados, que não são dados de localização, são todos do mesmo tipo. Exemplos deste tipo de dados são a densidade populacional, precipitação anual ou taxas de incidência de cancro.

2. **bivariate**(duas variáveis visuais), apresenta a distribuição geográfica de dois conjuntos de dados distintos. É exemplo de mapeamento bivariate, um mapa que mostre em conjunto a precipitação anual e a taxa de incidência de cancro, tentando desta forma explorar uma possível correlação entre os dois fenómenos.

3. **multivariate**(mais do que duas variáveis visuais), a única diferença para bivariate é que neste caso existem mais do que dois conjuntos de dados distintos.

Cada tipo de mapa temático permite representar os dados de forma diferente, sendo por este motivo usados para tarefas distintas, dependendo de quão adequados são para a tarefa que pretendermos. Entre os principais tipos de mapas temáticos, destacam-se:

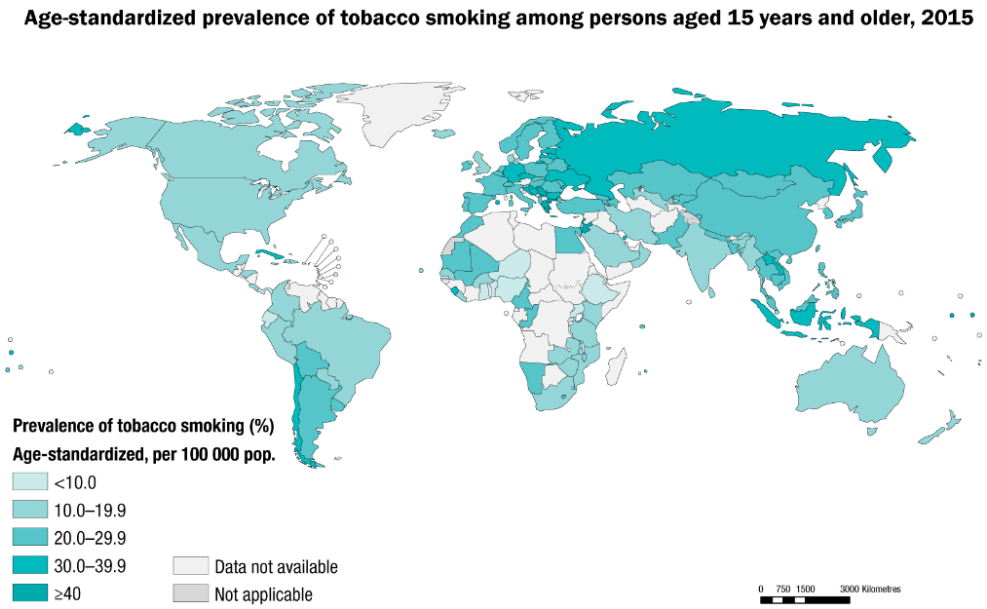


Figura 2.11: Mapa que representa a prevalência de fumadores acima dos 15 anos de idade nos vários países do mundo. Retirado de [3]

- **Mapas Choropleth:** os mapas do tipo Choropleth mostram dados que estão limitados geograficamente pelas regiões administrativas do mapa(tais como distritos, países) [31]. Este tipo de mapas assume uma distribuição uniforme dos fenómenos medidos em cada uma das regiões, sendo então impossível ao utilizador inferir dentro de uma área geográfica qual das suas zonas constituintes contribui mais ou

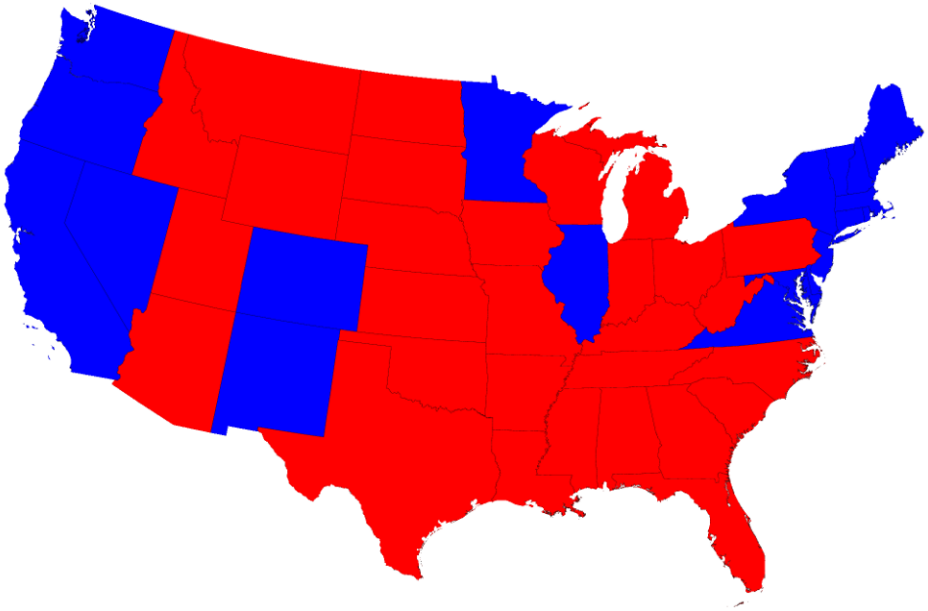


Figura 2.12: Mapa que representa os resultados das eleições de 2016 nos Estados Unidos por cada estado. Retirado de [43]

menos para o valor representado [50]. Nos mapas do tipo Choropleth usualmente cores mais saturadas são usadas para representar valores mais elevados do atributo. O mapa do tipo Choropleth da figura 2.11 mostra a prevalência uniformizada por idade, de fumadores com quinze ou mais anos de idade para o ano de 2015 em cada país do mundo. Nesta figura, países em que a percentagem de fumadores é maior, são representados por cores mais saturadas(mais escuras), ao passo que países em que a percentagem de fumadores é menor, apresenta uma cor menos saturada(mais clara). A figura usa um esquema de cores sequencial e a variável visual usada é a cor. É ainda de realçar que na legenda existem dois elementos que são usados para refletir países em que os dados não estão disponíveis (*Data not available*) ou o problema a ser mostrado não é aplicável (*Not aplicable*).

Na figura 2.12 é mostrado um exemplo de outro mapa do tipo Choropleth o qual usa um esquema de cores qualitativo. Neste mapa para cada estado é usada uma cor para representar qual a força política que ganhou as eleições de 2016, sendo que a cor vermelha representa vitória do partido Republicano e a cor azul a vitória do partido Democrata.

- **Mapas de Densidade de Pontos:** este tipo de mapas usam pontos para mostrar a presença de uma determinada característica ou fenómeno. Cada ponto corresponde a uma ocorrência ou conjunto de ocorrências. No caso de um ponto corresponder a uma só ocorrência então é necessário garantir que o ponto é representado na sua localização espacial correta. Quando um ponto representa várias ocorrências, os

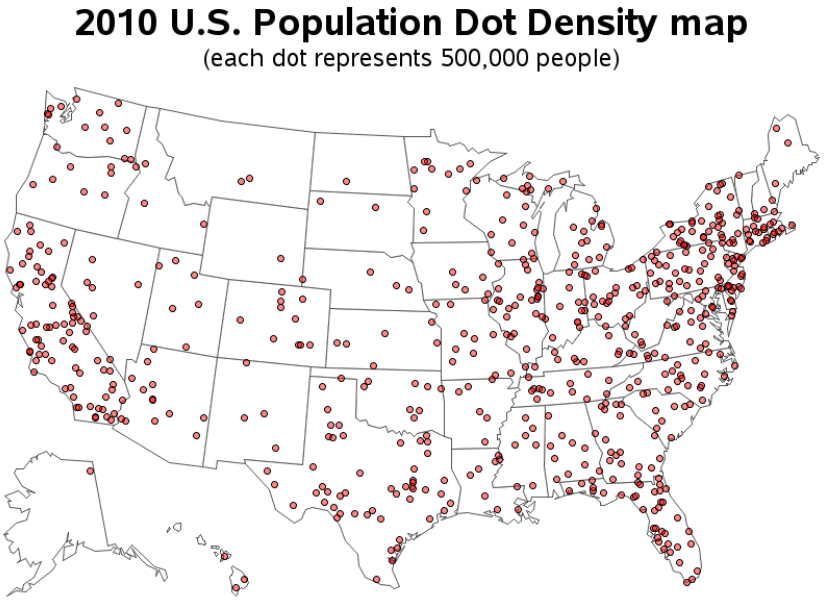


Figura 2.13: Mapa de densidade de pontos que efetua a contagem de pessoas nos Estados Unidos no ano 2010. Cada ponto equivale a 500 mil pessoas. Retirado de [1].

pontos representam dados agregados e a localização é arbitrária dentro da localização das ocorrências constituintes. Neste tipo de mapas é normalmente preciso tentar várias combinações para a quantidade e tamanho dos pontos para ver qual das combinações mostra da melhor forma os possíveis padrões nos dados [45].

Na figura 2.13 é mostrado um exemplo de um mapa de pontos em que cada ponto representa 500.000 ocorrências(pessoas). Através deste mapa em particular é possível identificar quais os estados que têm maior densidade populacional bem como quais os estados que têm menor densidade populacional.

Na figura 2.14 é possível ver outro tipo de mapa de densidade de pontos, um mapa em que cada ponto corresponde a uma ocorrência, neste caso uma pessoa que viva em Inglaterra ou no País de Gales. Neste mapa em que cada ocorrência corresponde a um ponto é fácil de ver as zonas de maior densidade populacional (zonas com grande quantidade de pontos muito juntos).

Tanto na figura 2.13 como na figura 2.14 é possível ver que alguns pontos se sobrepõe, o que leva a uma redução de legibilidade porque a sobreposição de pontos interfere com qualquer tentativa de fazer contagem dos mesmos.

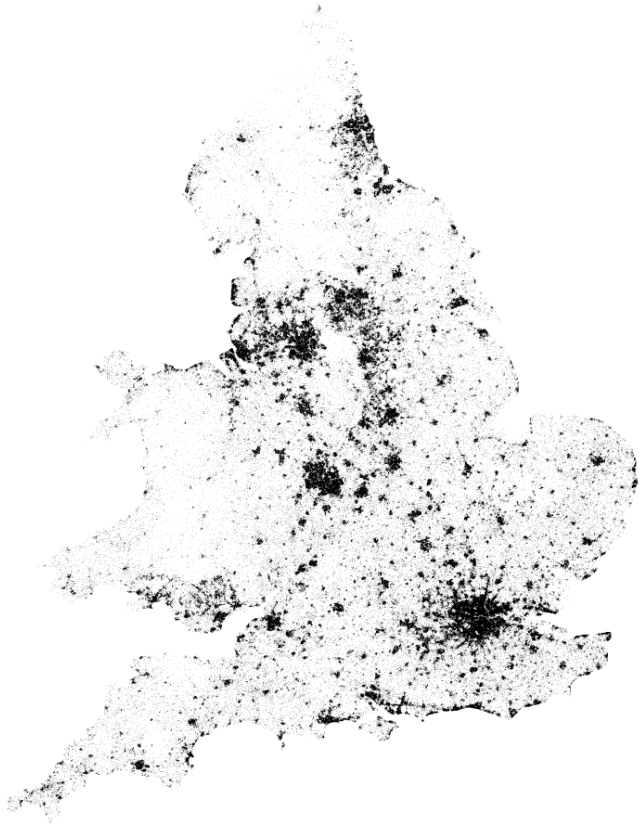


Figura 2.14: Mapa de densidade de pontos em que cada ponto mapeia uma pessoa que vive em Inglaterra e no País de Gales segundo os censos de 2011. Retirado de [21].

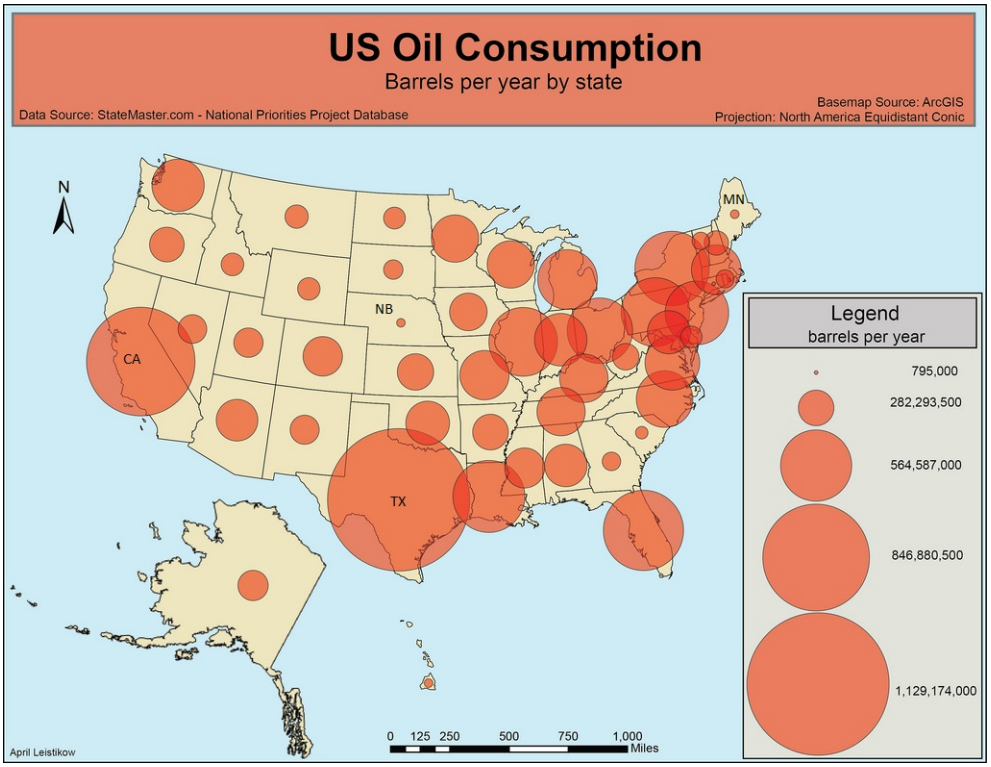


Figura 2.15: Mapa de símbolos proporcionais para o consumo de petróleo nos Estados Unidos, em barris por ano para cada estado. Retirado de [70]

- **Mapas de Símbolos Proporcionais:** os mapas de símbolos proporcionais usam símbolos de diferentes tamanhos para representar diferenças proporcionais dos dados associados a cada área ou localizações do mapa [31]. Os mapas de símbolos proporcionais podem ser usados com dados numéricos(e.g., altura de uma pessoa) ou dados categóricos ordenados(e.g., baixo, médio ou alto). São ainda flexíveis porque podemos usá-los para dados anexados a pontos geográficos(e.g., localização precisa) ou dados anexados a áreas geográficas (e.g., distritos, países). Uma vantagem deste tipo de mapas em relação a mapas de densidade de pontos é que é normalmente mais fácil estimar o tamanho de um símbolo do que contar o número de pontos existentes. Estes mapas usam as variáveis visuais forma e tamanho dos símbolos.

  A figura 2.15 representa um mapa de símbolos proporcionais para o consumo de petróleo em cada estado dos Estados Unidos, em barris por ano. Estados que têm um maior consumo de barris por ano têm um círculo de maior diâmetro(e.g., Texas e Califórnia), enquanto que estados que consomem menos barris por ano são representados com um círculo de menor dimensão (e.g., Nebraska e Maine). Contudo, quando temos símbolos que se sobrepõe e as zonas geográficas têm tamanho reduzido em comparação com os símbolos, torna-se difícil distinguir a zona geográfica associada a cada símbolo, sendo portanto necessário criar formas de desenho que possibilitem ao utilizador determinar o tamanho relativo dos símbolos de forma correta [16].

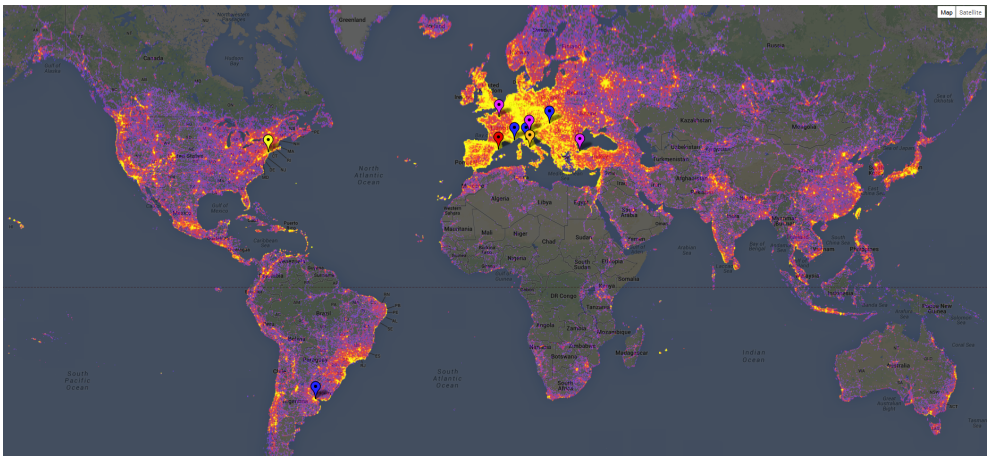


Figura 2.16: Heat Map que mostra os zonas com maior densidade de fotografadas tiradas na superfície terrestre. Retirado de [62]

- **Heat Map:** os Heat Maps representam valores de atributos quantitativos através do uso da cor, sendo que ao contrário dos Choropleths, nos Heat Maps os valores estão associados a coordenadas geográficas e não a áreas. São usados para representar fenómenos contínuos, utilizando para tal pontos e da interpolação do valor do fenómeno em cada ponto e da sua vizinhança resultam as cores finais.

Este tipo de mapa teve a sua origem no ano de 1991 pelas mãos de *Cormac Kinney*. *Cormac Kinney* usou os Heat Maps para representar informações do mercado financeiro em tempo real.

Na figura 2.16 pode ser visto um exemplo de um Heat Map que é usado para evidenciar os sítios que as pessoas mais visitam, baseado no número de fotos tiradas em cada um dos sítios. As zonas escuras representam locais onde existem poucas fotos tiradas, zonas a vermelho representam locais que têm mais fotografias tiradas do que as zonas escuras e por fim as zonas a amarelo são as zonas nas quais existem um maior número de fotografias tiradas.

- **Cartograma:** O cartograma é um mapa distorcido e cuja distorção resulta da substituição do valor da dimensão geográfica pelo valor do fenómeno em estudo.

  Os cartogramas podem ser de três tipos distintos: contínuos, não contínuos ou Dorling. Os cartogramas não contínuos permitem que as áreas geográficas possam ser separadas das áreas adjacentes permitindo aos objetos expandir ou contrair sem que haja deformação das áreas geográficas originais. Cartogramas contínuos, ao contrário dos não contínuos, não permitem que as áreas geográficas possam ser separadas das áreas adjacentes resultando em distorções da forma geográfica. Por fim, nos cartogramas do tipo Dorling, as áreas geográficas são substituídas por outra forma uniforme sem sobreposição, tal como um círculo, onde a área do círculo é proporcional ao atributo a ser representado [44]. Na figura 2.17 é possível visualizar os três tipos de cartogramas que existem, sendo que cada um à sua maneira, mostra a população por cada condado da Califórnia.

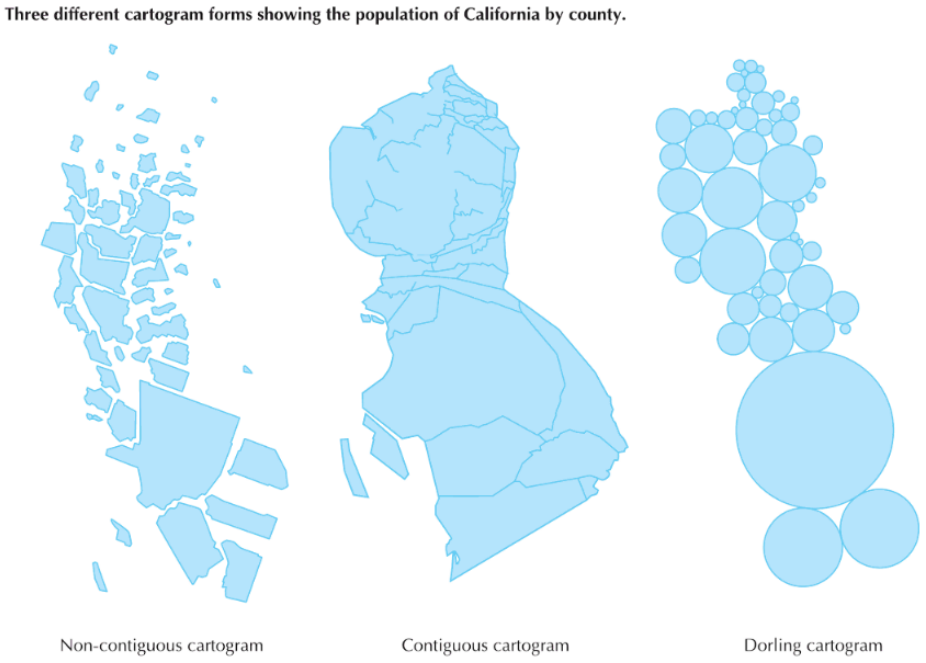


Figura 2.17: Tipos de Cartogramas. Retirado de [44]

Na figura 2.18 é possível ver um cartograma contínuo em que mostra os resultados das eleições norte americanas em 2008, no qual a cor vermelha representa um estado com uma maioria republicana e a cor azul representa um estado com uma maioria democrata. Neste caso em particular, a área foi distorcida proporcionalmente de acordo com o número de habitantes.

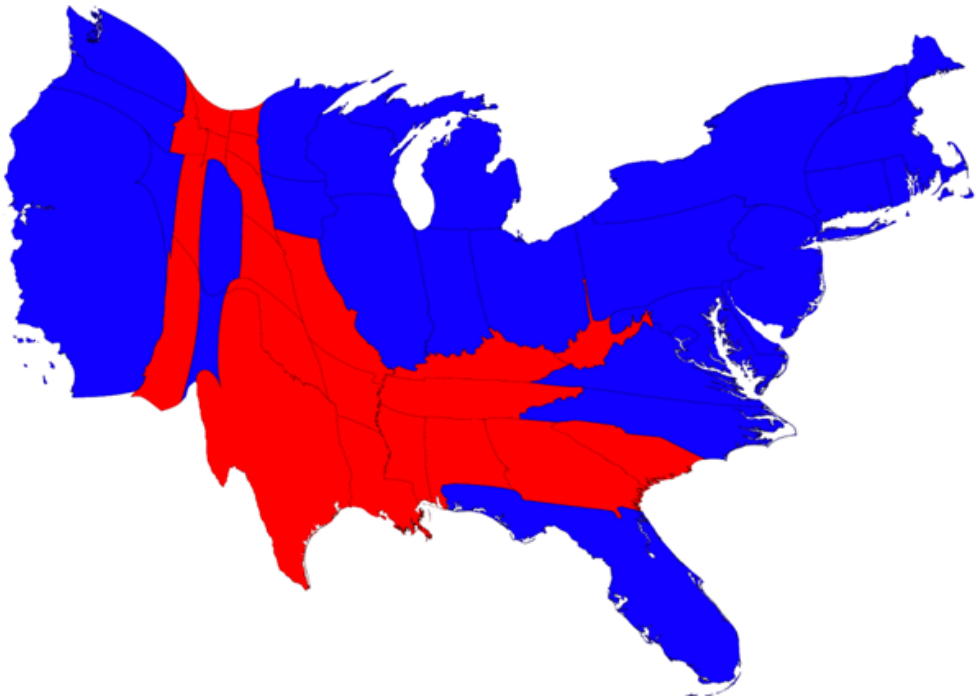


Figura 2.18: Cartograma contínuo que apresenta os resultados das eleições norte americanas em 2008. Retirado de [48]

Na figura 2.19 está disposto um cartograma não contínuo, sendo que usa duas variáveis visuais para transmitir a informação ao utilizador. Primeiramente, o tamanho de cada país é desenhado proporcionalmente à população do mesmo e depois a cor que é usada para o preenchimento dá-nos uma ideia da esperança média de vida, no qual cores mais escuras representam uma esperança média de vida mais baixa. Neste mapa, países populosos como a Índia e o Japão, ficam maiores do que a sua área geográfica normal, enquanto que países como a Rússia e a Mongólia, vêm o seu tamanho reduzido em relação ao tamanho normal.

Os cartogramas têm alguns problemas porque distorcem a geografia, implicando que as distâncias entre sítios não sejam precisas, sendo assim este tipo de mapas não pode ser corretamente interpretado, a não ser que o leitor conheça a geografia real do mapa.

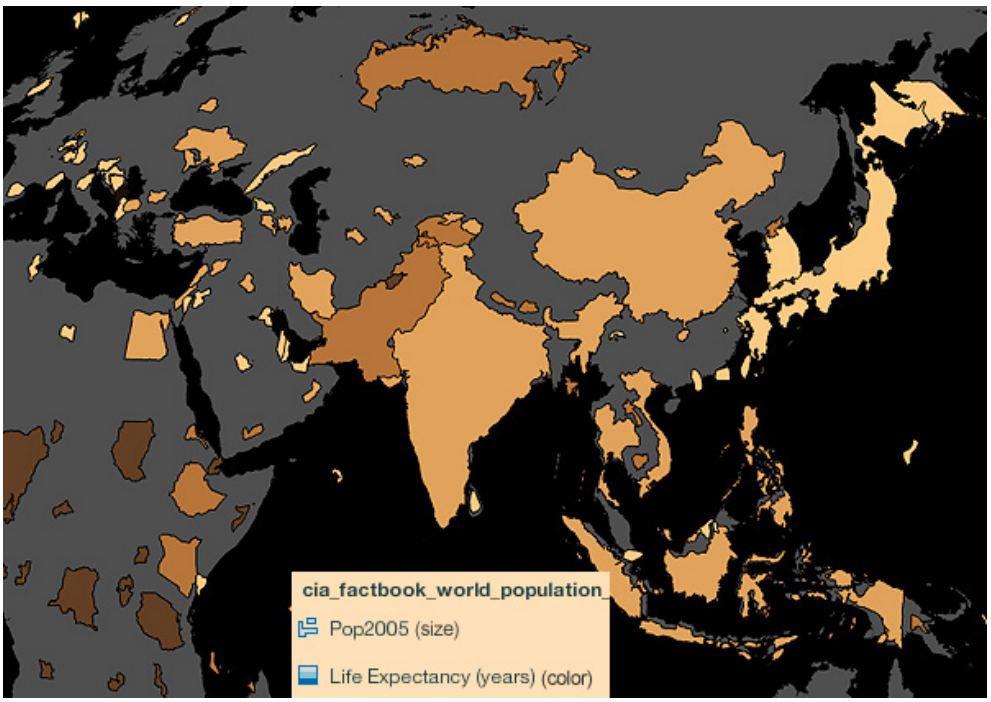


Figura 2.19: Cartograma não contínuo com duas variáveis visuais. Retirado de [65]

2.2.6 Mapeamento temporal

Como esta dissertação pretende adicionar a componente temporal a uma API de mapas temáticos previamente desenvolvida, existe a necessidade de conhecer os tipos de mapas temáticos existentes na literatura com o mapeamento temporal.

Essencialmente existem duas formas de mapear o tempo: (1) Mapear tempo para espaço e (2) Mapear tempo para tempo. O mapeamento de tempo para espaço consiste em representar o tempo e os dados na mesma representação visual, sendo que este tipo de visualização não muda ao longo do tempo, pelo que se costumam chamar de visualizações estáticas. Por outro lado, representações dinâmicas utilizam a dimensão física do tempo para transmitir a dependência temporal dos dados, isto é, o tempo é mapeado para o tempo. Isto resulta em visualizações que mudam ao longo to tempo (e.g., slide shows ou animações) [5, p.76].

De seguida irão ser mostradas várias formas para apresentar a componente temporal em mapas temáticos.

2.2.6.1 Animação

A animação impõe ao utilizador algumas exigências em relação ao mapeamento estático. Estas exigências são: (a) aumento da quantidade de informação a transmitir pelas diferentes frames da animação (comparada com uma única frame da visualização estática), (b) o tempo limite em que a informação dada por uma frame está disponível para ser vista, (c)

a necessidade de atender e relacionar mudanças que ocorrem simultaneamente em diferentes regiões da representação visual e (d) a necessidade de memorizar informação para integrar e relacionar com outra informação presente em frames diferentes da animação [39].

As animações funcionam bem sob o ponto de vista cognitivo se houver uma evolução em que é possível percecionar alguma variação espacial. Isto é, se como resultado da evolução existir da parte de quem está a fazer a interpretação da imagem, perceção de que existe movimento (pode ser apenas um padrão que existe naquela zona específica e que se está a deslocar), então nós humanos somos eficazes a detetar estas mudanças nos padrões. Sempre que isto não acontece e não há continuidade entre as frames (imagens) da animação, ou seja, que não nos dá nenhuma perceção de evolução do movimento então os humanos retiram poucas ilações, pois não conseguem interpretar corretamente este tipo de animações.

Apesar deste tipo de visualização de fenómenos ser visualmente apelativa, muitos dos modelos que usam animação são difíceis de compreender e não seriam superiores às representações estáticas porque a exigência de processamento de informação envolvidas nas animações pode ter efeitos negativos na capacidade de retenção de informação por parte do utilizador [39].

2.2.6.2 Change Map

Os mapas temáticos do tipo Change Map [7] são usados para estimar mudanças nas propriedades temáticas expressas por atributos numéricos entre períodos temporais. Além de estimar mudanças ocorridas em objetos individuais ou locais, este tipo de mapas permite, a quem está a analisar o mapa, detetar mudanças nos padrões gerais do mapa. Na figura 2.20 podemos visualizar um mapa do tipo Change Map que mostra as diferenças nas taxas de desemprego entre o ano de 1983 e 1990 nas várias divisões geográficas de Itália. Neste caso, é possível ver um padrão espacial nas mudanças: aumento da taxa de desemprego no Sul do país e diminuição no Norte. Nesta figura a cor amarela é usada para mapear variações positivas na taxa de desemprego e a cor azul é usada para mapear variações negativas da taxa de desemprego, isto é, usa um esquema de cores divergente.

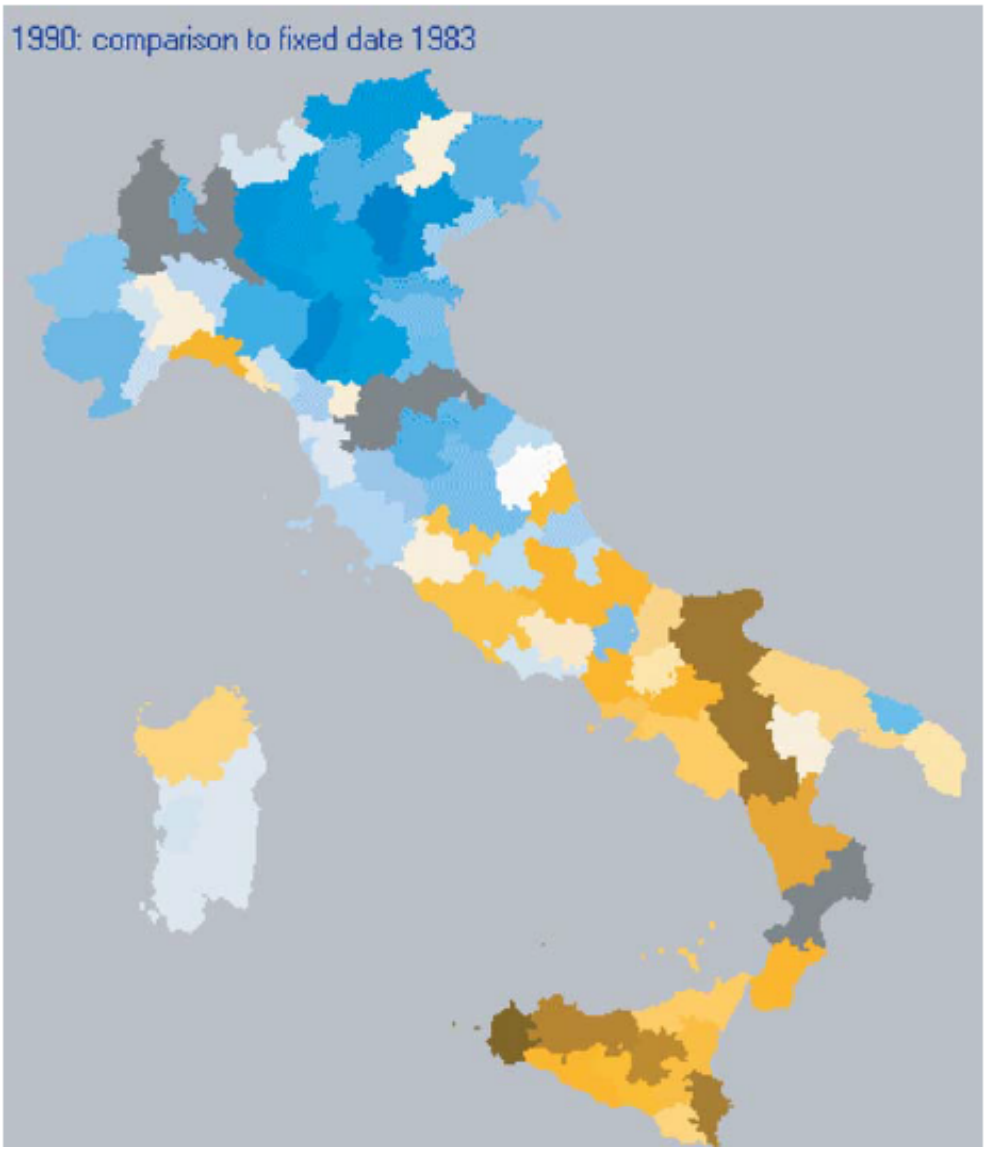


Figura 2.20: Change Map onde está representada a variação da taxa de desemprego em Itália no período de 1983 a 1990. Retirado de [7]

2.2.6.3 Space-Time Cube

Uma das técnicas de mapeamento temporal mais relevante na literatura é o Space-Time Cube, criado por *Torsten Hägerstrand*[29]. Esta técnica foi criada pela necessidade que existia em entender o que significava uma localização ter não só coordenadas espaciais mas também coordenadas temporais. Sendo assim, é uma técnica de visualização a três dimensões, que permite visualizar o espaço e o tempo simultaneamente.

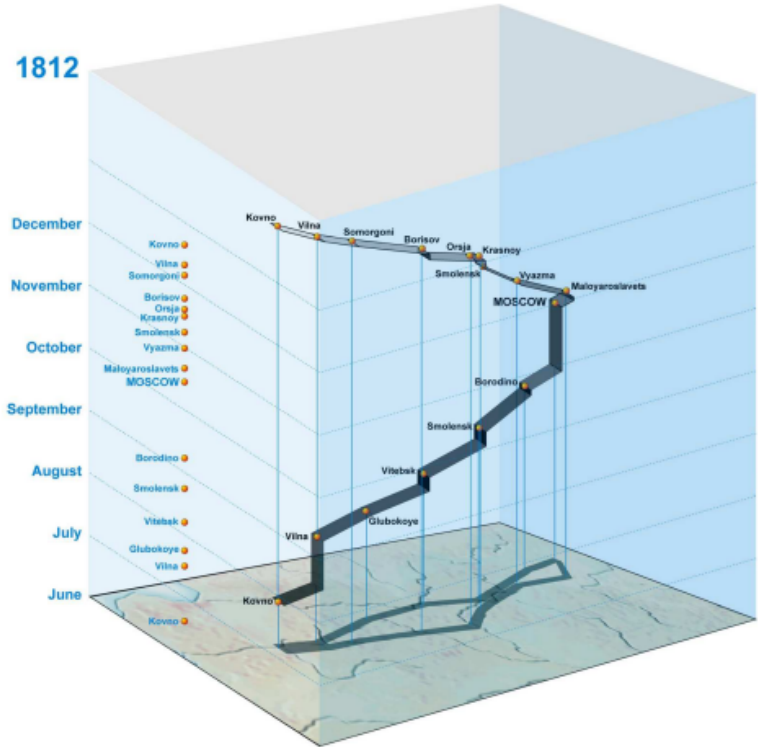


Figura 2.21: Space-Time Cube que permite visualizar a marcha de Napoleão na Rússia em 1812. Retirado de [49].

A ideia básica é mapear duas dimensões espaciais para dois eixos do cubo e usar o terceiro eixo livre para fazer o mapemento do tempo.

Um space-time cube consiste num cubo que, por exemplo, tem na sua base (ao longo do eixos dos x e y) a localização geográfica do fenómeno e cuja altura (eixo dos z) representa o tempo, de resolução variável (anos, dias, horas, etc), e que pode ser representado de forma contínua ou discreta. Pontos localizados mais perto da base representam ocorrências mais antigas do que pontos que estão mais afastados dela (mais acima). Na figura 2.21 é possível ver a marcha das tropas de Napoleão na Rússia ao longo do ano de 1812.

Ainda assim, este tipo de representação sofre de vários problemas: para grandes conjuntos de dados é muito difícil representar tudo de forma eficaz, sendo que neste tipo de representação, ao contrário das técnicas a duas dimensões, a informação tem de ser disposta nas faces do cubo e no seu interior.

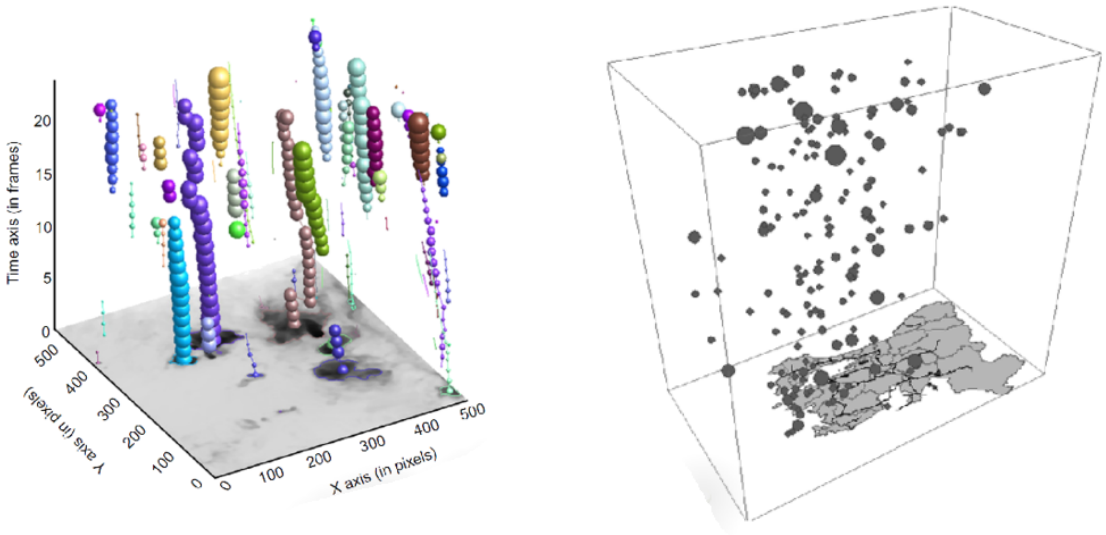
A técnica Space-Time Cube usualmente requer uma interação apropriada, para permitir aos utilizadores visualizar os dados em diferentes perspetivas, isto é, o cubo tem de ser manipulado no espaço para se obter o melhor ângulo de visualização sobre os dados em estudo. Estes problemas estão acima de tudo relacionados com as questões levantadas pelo uso de visualização a três dimensões.

Em [4, 5] é possível encontrar mais um conjunto de técnicas a três dimensões, as quais padecem dos mesmos problemas do Space-Time Cube ou de problemas semelhantes,

visto que necessitam de três dimensões para visualizar corretamente os dados e são todas técnicas que tiveram origem no Space-Time Cube.

Estas técnicas são:

1. **Spatio-Temporal Event Visualization** [5] - para analisar eventos no seu contexto espacial e temporal, podemos usar o conceito do space-time cube. Os eventos em si são visualizados ao colocarmos objetos no cubo, na posição espacial e temporal que os mesmos ocorrem. Atributos associados a cada evento podem ser codificados, fazendo variar a cor, forma, tamanho ou textura dos objetos. Nas figuras 2.22a e 2.22b é possível visualizar exemplos desta técnica. Na primeira figura, temos eventos relacionados com formação de nuvens de convecção e na segunda figura, os eventos representados são sismos.



(a) Nuvens de convecção

(b) Sismos

Figura 2.22: A técnica de visualização Spatio-Temporal Event Visualization. Retirado de [5].

2. **Time Varying Hierarchies on Maps** [5, p.243] - Estruturas hierárquicas podem ser encontradas em muitas áreas. A técnica Time Varying Hierarchies on Maps permite visualizar hierarquias que mudam ao longo do tempo num contexto geo-espacial. Esta técnica usa a terceira dimensão espacial para representar o tempo, o que é bastante parecido com a técnica space-time cube. Então, para cada espaço temporal, é construída uma camada(um novo mapa) disposta acima da camada respeitante ao espaço temporal anterior. Na figura 2.23 é possível visualizar um exemplo desta técnica, na qual cada camada(mapa) corresponde a um momento temporal onde são incorporadas as hierarquias. As diferenças nas cores entre camadas representam mudanças significativas do valor dos dados.

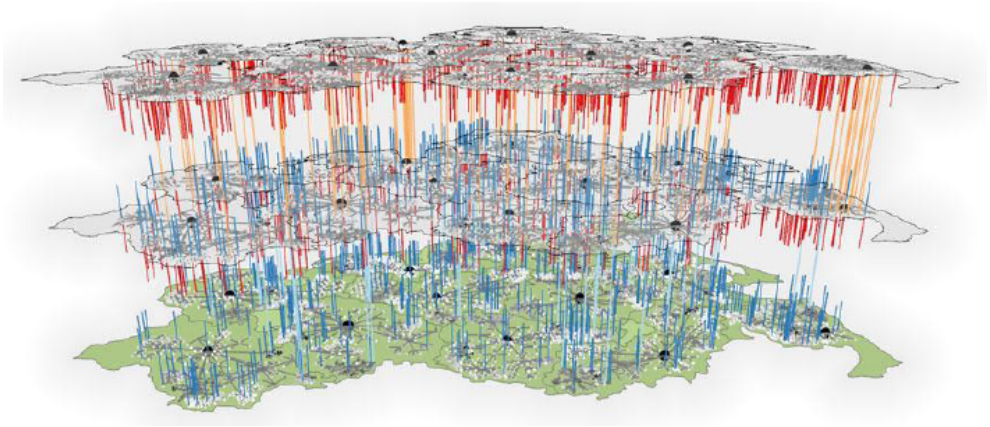


Figura 2.23: A técnica de visualização Time Varying Hierarchies on Maps. Retirado de [5, p.243].

3. **Geotime** [20] - é um sistema para visualizar itens (e.g., objetos, eventos, transações) no seu contexto espacial e temporal. É uma versão dinâmica, interativa do conceito space-time cube. GeoTime foi desenhado para melhorar a perceção e compreensão de movimentos de entidades, eventos, relações e interações com o decorrer do tempo dentro de um contexto geográfico e espacial. Os eventos são representados em 3D, usando dois eixos para a representação espacial e o terceiro para a componente temporal. Estes eventos são animados no tempo através do espaço em três dimensões quando movemos uma barra deslizante.

4. **Pencil Icons** [68] - é uma técnica que é usada para visualizar dados espácio-temporais com múltiplas variáveis e para tempo linear. Esta é outra técnica baseada no space-time cube na qual o mapa representa a componente espacial e o lápis a componente temporal. O lápis é usado como uma metáfora para permitir visualizar a mudança do tempo ao andar verticalmente no lápis, começando pela ponta. Cada face do lápis representa uma variável dependente do tempo. A cor é usada para visualizar os dados sendo que a sua variação depende do tipo de dados que forem usados, tendo o estudo de que tipos de cores se devem usar para cada tipo de variável feito anteriormente nesta dissertação. Na figura 2.24 é possível observar um exemplo desta técnica em que o tempo é representado pela parte vertical do lápis e em cada face do mesmo é representada uma variável dependente do tempo.

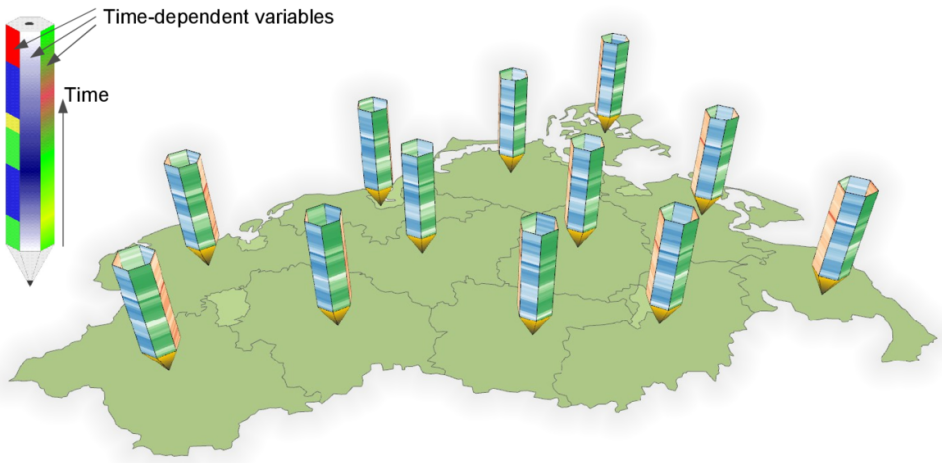


Figura 2.24: A técnica de visualização Pencil Icons. Retirado de [5].

5. **Helix Icons**[68] - é uma técnica muito parecida com o Pencil Icons, mas ao contrário dos Pencil Icons, os Helix Icons são úteis para destacar dados espácio-temporais com caraterísticas cíclicas. A diferença para os Pencil Icons é que para representar o domínio temporal é usada uma hélice em forma de fita em vez de um lápis. Cada segmento da fita em hélice permite visualizar um instante(ou intervalo) no tempo através de codificação usando cores.

2.2.6.4 Outras técnicas 3D

Na literatura referida na subsecção anterior existem ainda outras técnicas a três dimensões que importa referir, embora não sejam da maior utilidade para a realização desta dissertação. Os conceitos inerentes presentes nas mesmas podem ajudar e vir a ser usados mais à frente na solução final desta dissertação.

A primeira técnica a ter em conta são os **Data Vases** que, ao contrário do Space-Time Cube, possuem duas versões diferentes, uma a duas dimensões e outra a três dimensões, tendo sido uma técnica desenhada para permitir visualizar a variação de múltiplas variáveis temporais [63].

Um data vase em 2D é basicamente um gráfico em que o tempo é apresentado nas ordenadas. Nas abcissas para cada grânulo espacial tem um vaso que tem uma espessura conforme o aumento ou diminuição do valor da variável dependente do tempo. É fácil ver em quais grânulos espaciais e em que momento no tempo o valor da variável em estudo tem maior ou menor valor, através da sua largura. A duas dimensões, esta técnica fornece uma representação compacta das relações espácio-temporais nos dados. Ainda assim, este tipo de gráficos contém demasiada informação, e como tal precisa de uma boa legenda. Se tal não acontecer será muito difícil. Na figura 2.25 é possível observar os crimes violentos *per capita* cometidos na Carolina do Norte desde 1980 a 2006 através de um data vase 2D.

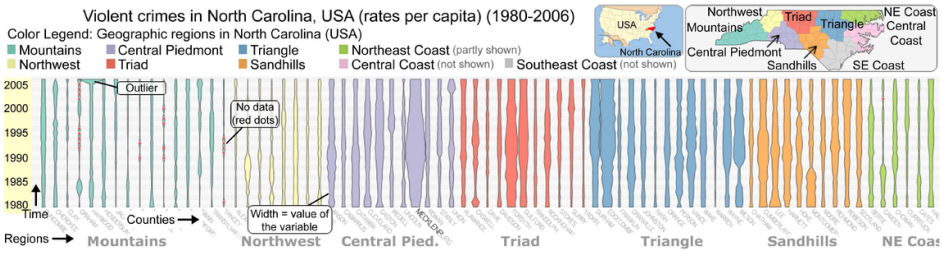


Figura 2.25: A técnica de visualização Data Vases 2D. Retirado de [5].

Nos Data Vases 3D, no centro de cada área espacial, os dados são representados na forma de um vaso. Para tal, é feito o mapeamento dos dados através de cor e largura enquanto que o eixo temporal é representado na vertical. Cada passo temporal(disco), tem maior diâmetro quanto maior for o valor da variável representada. A cor em cada disco temporal codifica valores dos dados e quantidades estatísticas. Esta é muito importante para expor valores pouco usuais nos dados, como "outliers" e mudanças abruptas, especialmente quando o conjunto de dados é muito denso. Na figura 2.26 é possível observar as taxas de desemprego em termos médios mensais na Carolina do Norte desde 1999 a 2008 através de um data vase 3D.

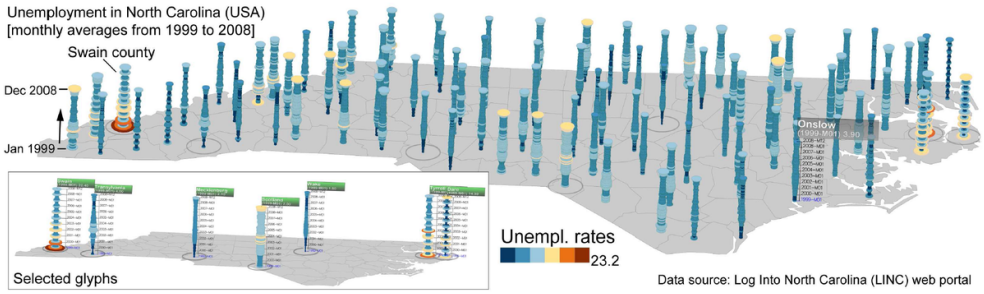


Figura 2.26: A técnica de visualização Data Vases 3D. Retirado de [5].

Nenhuma das versões desta técnica é ideal para o trabalho relacionado com esta dissertação. Na versão a duas dimensões, a informação é disposta num gráfico, não num mapa, e a versão a três dimensões não é ideal porque acarreta dificuldades tanto a nível da implementação como de utilização.

Outra técnica a três dimensões é a **Great Wall of Space Time** [67]. Nesta técnica, o espaço geográfico tem duas dimensões e o tempo uma dimensão linear. A construção da muralha que a técnica usa é baseada em aspetos topológicos e geométricos do espaço geográfico. O objetivo da muralha não é mostrar todas as áreas geográficas, mas pelo contrário, focar-mo-nos em fatias dessa área geográfica. A muralha age como uma tela na qual se podem projetar visualmente representações do espaço e dados dependentes do tempo. A dimensão temporal é mapeada ao longo da extensão vertical da muralha e usam-se cores para codificar diferentes valores que os dados possuam. Na figura 2.27 é possível observar um exemplo desta técnica.

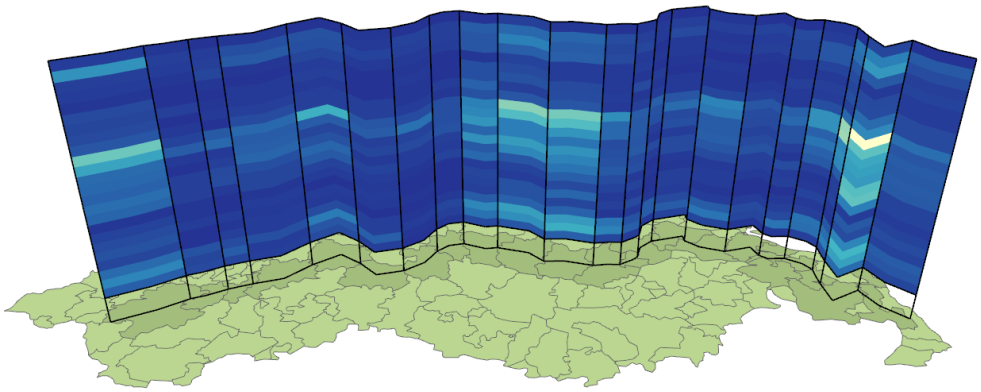


Figura 2.27: A técnica de visualização Great Wall of Space Time. Retirado de [67].

Na técnica **Trajectory Wall** [69] é possível ver aspetos temporais, espaciais e de atributos de dados relativos a movimentos, sendo uma representação híbrida(2D/3D). Primeiro agrupam-se as trajetórias adequadamente, depois empilham-se as trajetórias em cima do mapa. Um atributo associado a trajetórias(e.g. velocidade, aceleração) é depois selecionado pelo utilizador e é codificado usando cor em cada uma das camadas existentes. Linhas em duas dimensões são adicionadas ao mapa para melhor mostrar o aspeto espacial do movimento. Esta é uma técnica complexa e que requer boa manipulação a três dimensões para ser possível extrair informação da mesma.

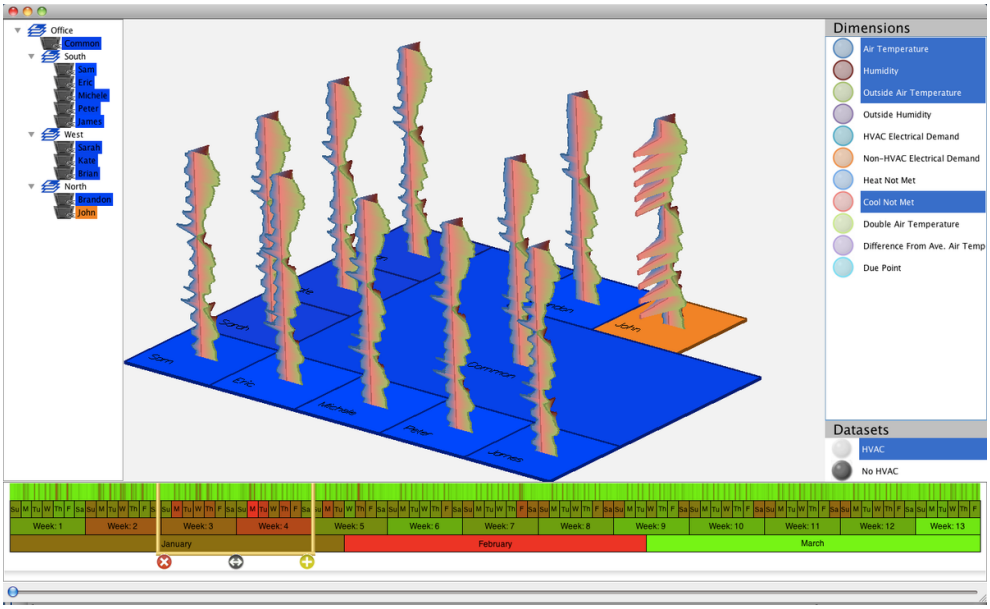


Figura 2.28: A técnica de visualização Wakame. Retirado de [22].

Para finalizar as técnicas de visualização que usam três dimensões temos o **Wakame** [22]. Esta técnica é um sistema interativo para visualizar dados espácio-temporias com múltiplas variáveis. Os aspetos temporais são codificados em "glifos", chamados Wakame. Um Wakame corresponde a um gráfico disperso ao longo da terceira dimensão. Esta dimensão vertical representa a variação temporal, onde os objetos são construídos a partir

da linha central e permitem ver tendências e relações entre as variáveis dependentes do tempo. Quando temos múltiplos Wakame num mapa torna-se mais fácil a compreensão dos aspetos temporais. Usando esta técnica de visualização e um bom posicionamento do ângulo de visualização podemos obter resultados interessantes. Na figura 2.28 é possível observar um Wakame para cada localização de um sensor.

2.2.6.5 Técnicas 2D

Já vimos as várias técnicas existentes na literatura para três dimensões e quais as razões que nos levam a não considerá-las para a realização desta dissertação, importando agora verificar as técnicas de visualização que usem duas dimensões.

A primeira técnica são os **Growth Ring Maps**[10], que é uma técnica para visualizar a distribuição espácio-temporal de eventos. Cada evento espácio-temporal é representado por um pixel, o que torna a técnica bastante escalável com o aumento do número de eventos. Os eventos(pixeis) são agrupados a partir de um ponto central de forma orbital resultando nos chamados anéis de crescimento. Os pixeis são organizados por data e tempo da sua ocorrência: quanto mais cedo o evento aconteceu mais perto do ponto central, o mesmo se encontrará. Quando dois ou mais pontos se encontram no mesmo sítio, o algoritmo que faz a computação do local onde o evento iria ser colocado, calcula as posições de modo a que não existam dois pixeis sobrepostos. Quando existem grupos em que acontecem bastantes sobreposições a forma final do anel não será perfeitamente circular. Os anéis de crescimento resultantes deste processo são depois sobrepostos sobre um mapa cartográfico para capturar o seu contexto espacial.

Na figura 2.29 é possível visualizar um Growth Ring Map no qual está disposta a distribuição espácio-temporal de fotos tiradas na Suíça. Nos anéis em que existem várias cores indicam que foram tiradas fotos ao longo de vários anos e estações dos respetivos anos, enquanto que uma zona como por exemplo a marcada com a letra D teve muito mais fotografias tiradas no Outono do que nas outras estações do ano.

Esta técnica permite detetar padrões em grupos que possam existir nos dados recolhidos. Mas em contrapartida, pressupõe que as ocorrências tenham uma distribuição espacial com um elevado grau de concentração por área, tornando possível a formação de anéis em redor do centro. Devido a este pressuposto esta técnica funciona mal para conjuntos de dados dispersos.

A técnica de visualização **Icons on Maps** [23] é útil quando os dados orientados pelo tempo contém dependências espaciais. Nestes casos queremos visualizar os aspetos temporais e espaciais dos dados. Uma possível abordagem é adaptar soluções já existentes. Como os mapas são normalmente utilizados para representar o contexto espacial dos dados, então para representar a componente temporal num mapa, as técnicas de visualização terão de ser tornadas compatíveis com o mapa. Primeiro importa reduzir o tamanho da representação visual para um ícone e de seguida, colocar os vários ícones das representações visuais, no mapa. Cada um dos ícones vai ser colocado na sua região espacial

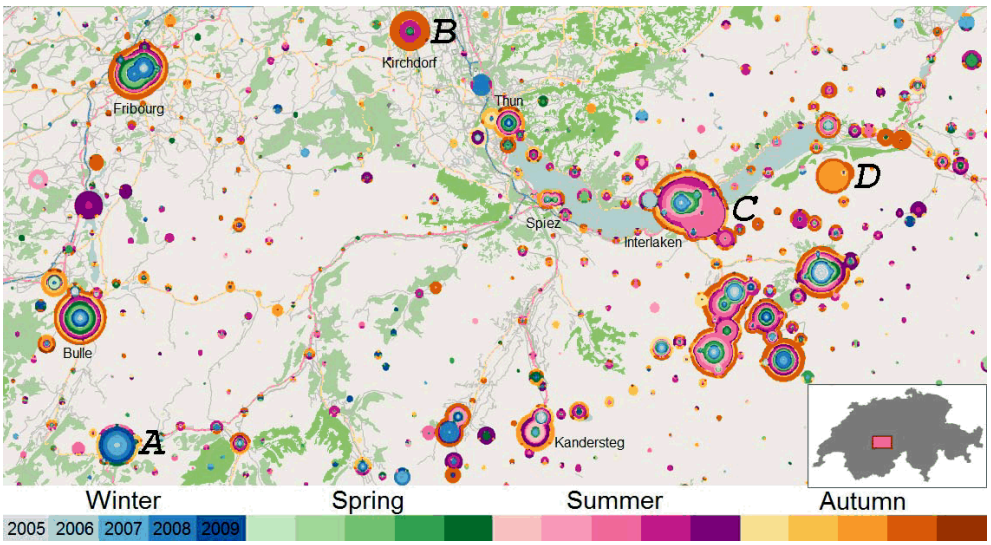


Figura 2.29: Growth Ring Maps que mostra a distribuição de imagens tiradas na Suíça. Retirado de [26]

respetiva.

A técnica icons on maps é de difícil compreensão, quando o mapa de fundo é pequeno, as regiões geográficas pouco extensas ou com muitos ícones para serem dispostos. É preciso ter em atenção que para cada ícone a sua representação terá uma região espacial particular que pode ser maior ou mais pequena, sendo isto uma limitação para o tamanho do ícone.

A técnica **Value Flow Map** [8] é uma técnica para visualizar variações em dados espácio-temporais. Este tipo de mapas mostra para cada zona do mapa cartográfico (e.g., distritos, países), um gráfico miniatura, que representa o comportamento temporal de uma variável. Para conseguir mostrar os dados num gráfico miniatura é necessário levar a cabo algumas operações sobre os mesmos dados para suavizar as variações. Por exemplo, substituir os valores de um ponto pela média de um intervalo de pontos na escala temporal. Desta maneira, pequenas flutuações são descartadas e as tendências principais ficam visíveis. Um exemplo possível deste tipo de transformações é mostrar o desvio da variável em relação à média dos dados, ou seja, os valores são substituídos pelas diferenças para a média, para desta forma representar variações positivas e negativas.

Na figura 2.30 é possível observar um Value Flow Map, no qual para cada estado dos Estados Unidos estão dispostas as estatísticas de crimes, neste caso o roubo de veículos motorizados. Para cada estado existe um gráfico que dá a ideia da diferença em relação à média, sendo a cor amarela para codificar um desvio positivo e a cor azul para codificar um desvio negativo.

Os **Flow Map** são usados para dados que estajam relacionados com movimento, isto é, registos sobre posições espaciais de quaisquer agentes em movimento(e.g., pessoas, veículos, animais) em diferentes momentos temporais. Ou seja, mostram a variação das posições no tempo, ao invés de variações nos valores dos dados [9].

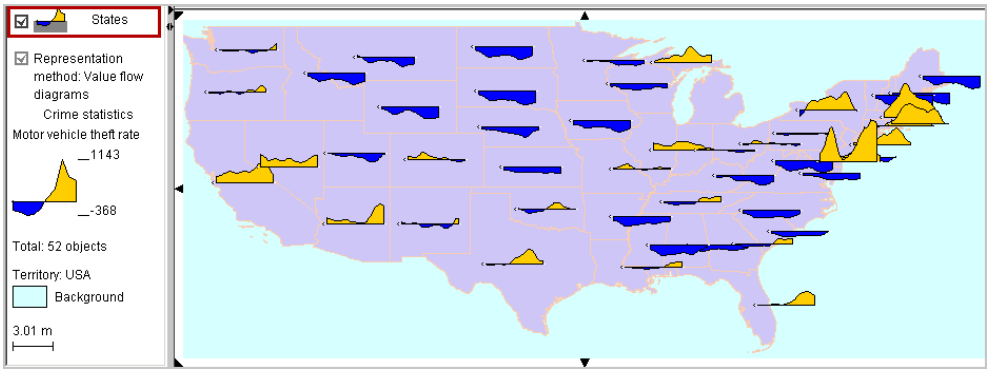


Figura 2.30: Value Flow Map. Para cada gráfico miniatura, a cor amarela é usada para codificar desvio positivo da variável em relação à média e a cor azul um desvio negativo. Retirado de [71]

Usualmente, os movimentos formam trajetórias diretas(opcionalmente segmentadas) que conectam o ponto inicial do movimento e o ponto final. Estas trajetórias podem ser representadas visualmente com setas ou curvas, mais ou menos complexas, onde a largura, cor e outros atributos podem ser usados para codificar informação adicional.

Para representar um grande número de fluxos é preciso sobrepor as trajetórias, ou seja, nos flow maps tem de se abstrair movimentos em grupo em vez de movimentos individuais. Primeiramente os pontos são agregados em intervalos de tempo mais largos e sítios individuais são substituídos por uma região mais larga(e.g., substituir dados de um edifício em particular da universidade pela localização da faculdade em si) sendo que desta forma temos trajetórias que mostram tendências médias. Depois as trajetórias são agrupadas por similaridade, e as que forem similares são agrupadas permitindo que trajetórias individuais sejam substituídas por trajetórias em grupos.

Esta técnica é bastante útil para visualizar trajetórias a duas dimensões, como podemos ver na figura 2.31, permitindo agrupar trajetórias e assim detetar padrões nos dados.

Os **Flowstrates** [14] estendem a ideia dos flow maps para a dimensão temporal e permitem ao utilizador analisar mudanças de magnitude ao longo do tempo.

Nos Flowstrates a origem e o destino dos fluxos são mostrados em dois mapas diferentes, e as mudanças da magnitude dos fluxos no tempo são mostradas entre os dois mapas numa espécie de "heatmap", no qual as colunas representam períodos temporais.

Na figura 2.32 é possível ver um exemplo desta técnica na qual podemos ver para o intervalo entre 1980 e 2009 quais foram os fluxos de refugiados do Sudão para vários países da Europa. Um problema deste tipo de mapas é as rotas entre os dois pontos, origem e destino, não serem corretamente representadas e ainda a elevada complexidade que a visualização terá de possuir.

De seguida, temos os **Time Oriented Polygons on Maps** [52], uma técnica que permite visualizar mudanças temporais dentro de cada polígono espacial que constitui o mapa.

Fatias, anéis ou pedaços temporais são as três formas usadas para mostrar mudanças

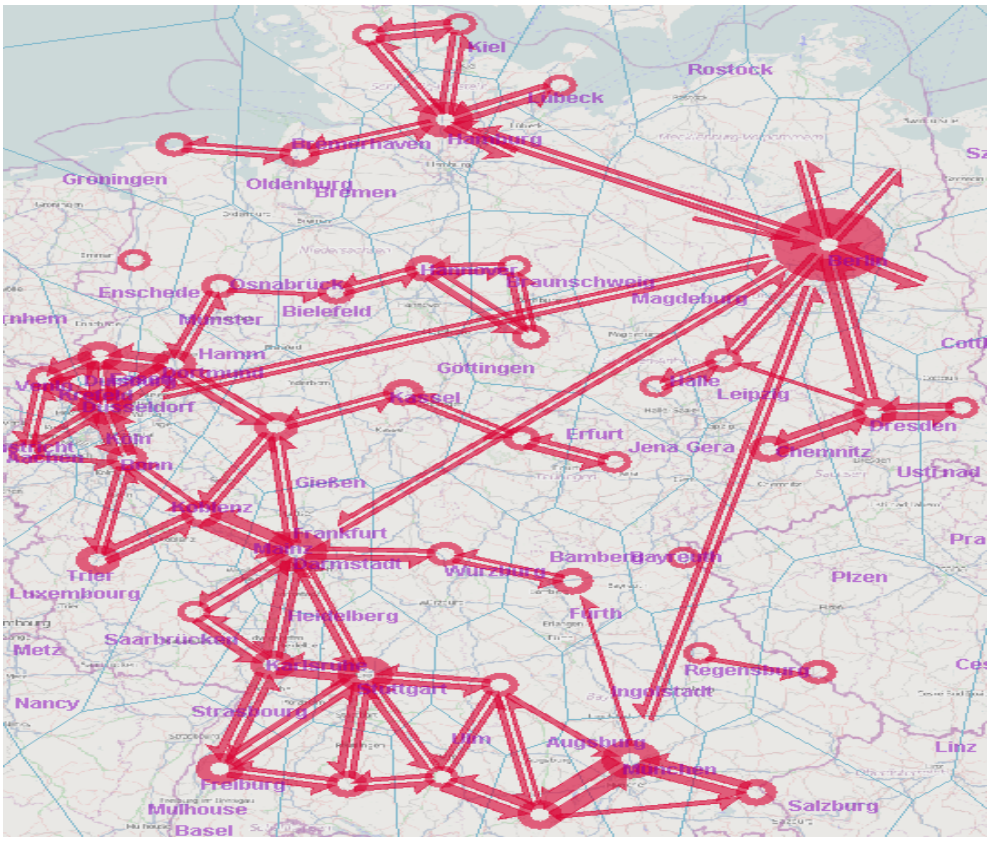


Figura 2.31: Flow Map que mostra os movimentos caraterísticos de fotógrafos ao longo do tempo. Informação extraida de metadados de mais de 590,000 fotografias geo-referenciadas. Retirado de [6]

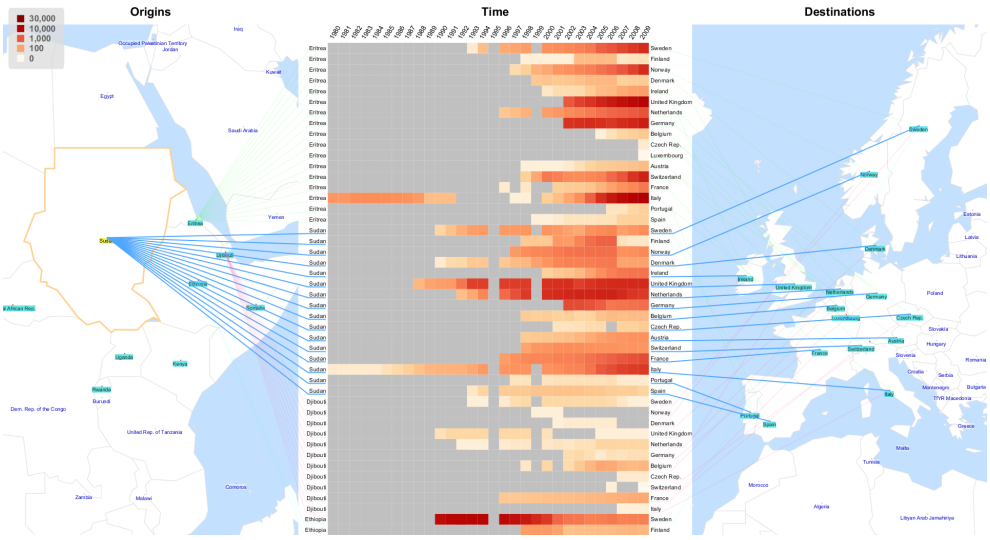


Figura 2.32: FlowStrates.Nesta imagem podemos ver os fluxos de refugiados para cada ano entre 1980 e 2009, desde o Sudão e vários países Europeus. Para cada ano a coluna respetiva tem uma cor associada que representa o número de refugiados que migrou da origem para o destino. Retirado de [13]

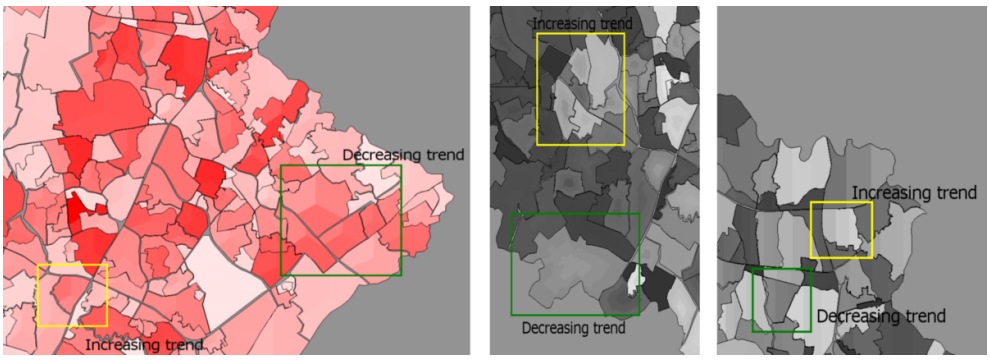


Figura 2.33: Time Oriented Polygons on Maps. Primeiro (a) Divisão em fatias temporais, depois (b) divisão em anéis e por fim (c) divisão em pedaços temporais. Retirado de [53]

nos valores dos dados com o passar do tempo. Para todas as variações, os valores dos dados e as categorias são representadas usando as várias componentes da cor (saturação, matiz e luminosidade).

Em mapas que usem polígonos divididos em fatias(figura 2.33(a)), a área dos polígonos é particionada de maneira semelhante a um relógio sendo que o número de setores é igual ao número de pontos temporais (divisões do tempo) existentes. Mapas que usem a divisão de polígonos por anéis (figura 2.33(b)), o tempo varia do centro para a parte mais exterior do polígono, correspondendo o centro ao tempo mais antigo. Em mapas que usem divisão de polígonos por pedaços temporais(figura 2.33(c)), cada pedaço vertical está ordenado temporalmente da esquerda para a direita.

Neste tipo de técnica se as divisões temporais forem muitas para serem visualizadas ao mesmo tempo, será muito difícil ao utilizador retirar informação pertinente do que estiver a visualizar. Como podemos ver na figura 2.33 existindo poucas divisões temporais é possível extrair alguma informação (neste caso aumentos e diminuições), que seriam de difícil análise caso existissem muitas divisões temporais.

Quando os dados espácio-temporais forem complexos e multi-facetados, costuma-se usar a técnica denominada **VIS-STAMP** [27] que integra métodos computacionais, visuais e cartográficos para explorar e analisar visualmente os dados espácio-temporais, que são na sua natureza complexos.

Sendo um método complexo, é melhor explicado através da figura 2.34, na qual podemos ver que este sistema de visualização é constituído por quatro componentes. No canto superior esquerdo é possível ver uma matriz em que cada coluna representa a variação do tempo e cada linha uma região geográfica. A afiliação a um cluster por parte das células da matriz é visualizada através de um esquema de cores que é consistente em todas as vistas do mapa.

No canto superior direito são mostradas várias vistas do mapa, em tamanho pequeno, uma para cada momento temporal. Sendo cada uma das miniaturas um mapa do tipo Choropleth, uma para cada momento temporal. No canto inferior esquerdo tem a vista chamada SOM(Self Organizing Maps) [35] que contém detalhes para controlar a interface.

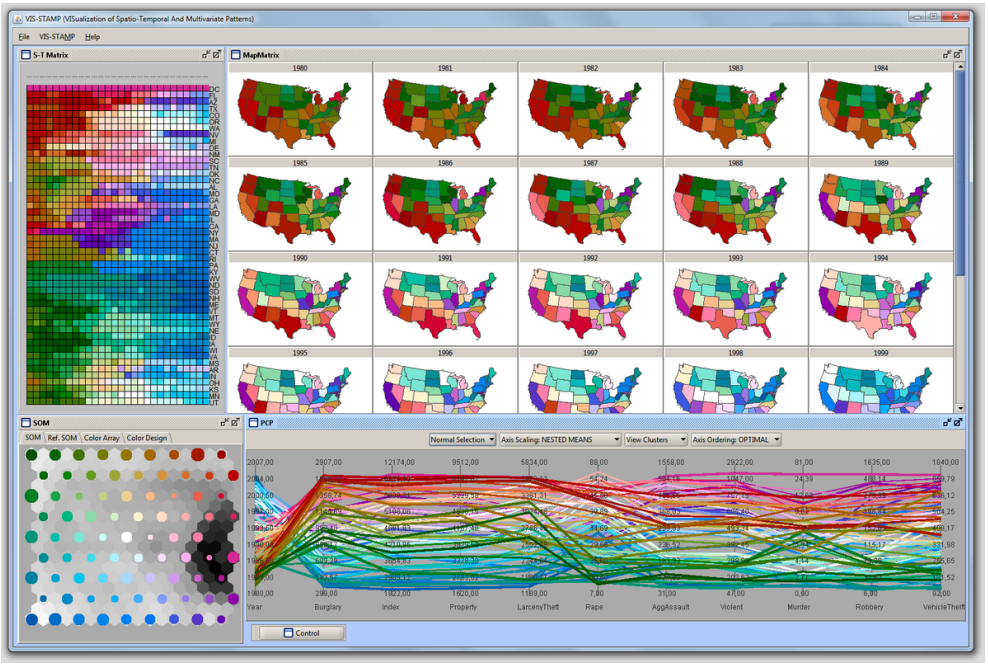


Figura 2.34: Exemplo da técnica VIS-STAMP. Retirado de [72]

Por fim, no canto inferior direito existe uma vista que permite lidar com as múltiplas variáveis que os dados possuem.

Esta é uma técnica complexa mas que pode ser eficaz permitindo mostrar vários momentos temporais na mesma visualização, sendo fácil verificar as diferenças entre cada momento temporal.

2.2.6.6 Resumo

Agora que já vimos um vasto conjunto de técnicas para mapeamento temporal é de todo o interesse apresentar um resumo das mesmas para facilitar a sua compreensão. Na tabela 2.35 é possível visualizar as diversas técnicas de mapeamento temporal que foram vistas ao longo desta secção. Primeiramente, para cada técnica, temos o número de variáveis visuais que cada técnica suporta. Técnicas que suportam uma única variável no domínio são conhecidas como univariadas e técnicas que suportem mais do que uma variável são conhecidas como multivariadas.

Depois temos o arranjo do domínio temporal, que corresponde à nossa perceção do tempo, dividindo-se em linear e cíclico. No arranjo linear o tempo é percecionado como precedendo linearmente do passado para o futuro, enquanto que, no arranjo cíclico o domínio temporal é composto por um conjunto de valores recorrentes (e.g., estações do ano). No arranjo é de referir que todas as técnicas lidam com arranjo linear enquanto que só cinco (Helix Icons, Trajetory Wall, Icons on Maps, Growth Ring Maps e Time Oriented Polygons on Maps) lidam com arranjo cíclico. A única técnica somente preparada para arranjo cíclico é a técnica Helix Icons pois foi pensada especialmente com essa finalidade.

	Variáveis Visuais		Arranjo		Primitivas Temporais		Mapeamento		Dimensionalidade	
	Univariado	Multivariado	Linear	Cíclico	Instante	Intervalo	Estático	Dinâmico	2D	3D
Change Map	✓		✓			✓	✓		✓	
Space-Time Cube	✓	✓	✓		✓	✓	✓			✓
Space-Time Path	✓	✓	✓		✓	✓	✓			✓
Spatio-Temporal Event Visualization	✓	✓	✓		✓		✓			✓
Time Varying Hierarchies on Maps	✓		✓		✓		✓			✓
Geotime		✓	✓		✓		✓	✓		✓
Pencil Icons		✓	✓		✓	✓	✓			✓
Helix Icons		✓		✓	✓	✓	✓			✓
Data Vases		✓	✓		✓	✓	✓		✓	✓
Great Wall of Space Time	✓		✓		✓		✓			✓
Trajetory Wall	✓		✓	✓	✓		✓		✓	✓
Wakame		✓	✓		✓		✓			✓
Growth Ring Maps	✓		✓	✓	✓		✓		✓	
Icons on Maps	✓	✓	✓	✓	✓	✓	✓		✓	✓
Value Flow Map	✓		✓		✓		✓		✓	
VIS-STAMP		✓	✓		✓		✓		✓	
Flowstrates	✓		✓		✓		✓		✓	
Flow Map	✓		✓		✓	✓	✓		✓	
Time Oriented Polygons on Maps	✓		✓	✓	✓		✓		✓	

Figura 2.35: Tabela de Mapeamento Temporal para os diversos tipos de técnicas existentes.

De seguida, temos as primitivas temporais que nos dizem como os dados relacionados com o tempo estão organizados. Estas primitivas estão divididas essencialmente em instante e intervalo. Um instante é um modelo para um ponto singular no tempo (e.g., 10 de Maio de 2016), ao passo que os intervalos são como o nome indica intervalos entre dois pontos no tempo (e.g., desde 10 de Maio de 2016 a 15 de Maio de 2016).

Posteriormente, temos o mapeamento, o qual se pode dividir em: (1) mapeamento de tempo para espaço ou estático e (2) mapeamento de tempo para tempo ou dinâmico. Pela tabela é possível visualizar que todas as técnicas possuem mapeamento estático, sendo que, apenas o GeoTime permite mapeamento dinâmico e estático.

Por fim, em termos de dimensionalidade, as técnicas dividem-se nas que trabalham a duas dimensões e a três dimensões. Neste tópico em particular existe um equilíbrio entre o número de técnicas que funcionam em cada dimensão. No geral, é de referir que as técnicas a três dimensões sofrem de vários problemas, nomeadamente problemas na elaboração e na necessidade de manipulação.

2.3 Mapas Temáticos na Web

Nesta secção vão ser introduzidas as plataformas web para mapas temáticos através da descrição das arquiteturas que são usadas para distribuir os dados e para os renderizar num mapa temático.

2.3.1 Plataformas Web para Mapas Temáticos

A construção de mapas temáticos na Web pode ser feita de duas formas diferentes: arquitetura cliente/servidor e arquitetura standalone. É importante perceber como cada uma pode ser usada para a construção de mapas temáticos na Web e quais as diferenças fundamentais entre as duas abordagens.

2.3.1.1 Arquitetura Cliente/Servidor

Na arquitetura do tipo cliente/servidor a geração do mapa temático é trabalho do servidor, sendo que o cliente só dispõe a informação recebida do servidor no referido mapa temático. Primeiramente, o cliente faz um pedido e recebe do servidor uma ou mais imagens para depois mostrar ao utilizador que está a analisar o mapa temático, sendo que estas imagens contêm o mapa de fundo juntamente com a camada temática. Outra possibilidade é o servidor enviar o mapa de fundo e a camada temática em imagens separadas, as quais são colocadas por ordem pelo cliente. Independentemente disso, quando o utilizador faz uma qualquer ação que mude o contexto espacial (pan ou zoom) e/ou o contexto temporal (através de um slider mudar o ano a visualizar no mapa temático), haverá um ou mais pedidos ao servidor para que este processe o novo contexto e gere as respetivas imagens e as envie para o cliente de forma a permitir que este as mostre ao utilizador.

Este tipo de arquitetura por um lado retira do dispositivo cliente o processamento e geração de novas imagens, que é feita pelo servidor, sendo assim necessário poucos requisitos de hardware para o cliente pois este só vai mostrar o resultado final. No entanto, leva a que o servidor tenha requisitos de hardware mais elevados, pois o servidor terá de tratar todos os pedidos e gerar todas as imagens e enviar as mesmas ao cliente que fez o pedido respetivo. É ainda, de ter em conta o tempo que resulta da necessidade de contactar o servidor quando o utilizador muda o contexto espacial ou temporal. Com o tempo que a informação leva entre o cliente e o servidor e o tempo de tratamento por parte do servidor, a interatividade é normalmente afetada de forma negativa.

Esta arquitetura requer da parte do programador um esforço adicional para configuração de serviços e também requer hardware potencialmente mais caro do lado do servidor para que o tempo de tratamento e geração de resposta a pedidos seja o menor possível.

2.3.1.2 Arquitetura Standalone

Na arquitetura standalone o cliente faz o pedido inicial e recebe da parte do servidor os dados necessários para a construção do mapa temático no seu contexto espacial e temporal inicial e/ou para os vários contextos espaciais e temporais que o mapa temático possua.

Nesta arquitetura o servidor tem requisitos de hardware inferiores em relação aos necessários para a arquitetura cliente/servidor porque ao contrário desta, na arquitetura standalone o servidor apenas recebe pedidos dos clientes e responde com os dados

correspondentes ao pedido que recebeu. Por outro lado, nesta arquitetura o cliente fica responsável pelo tratamento e disposição dos dados no mapa temático, tendo assim, requisitos de hardware superiores à arquitetura cliente/servidor ou de outra forma colocará em risco a interatividade do utilizador para com o mapa temático.

No caso da arquitetura standalone o servidor também é necessário, pois quando se dá um evento (pan ou zoom) que altere o contexto espacial, terá de ser feito um novo pedido para que sejam devolvidas as imagens necessárias. Neste caso o mapa de fundo é de extrema importância pois é ele que fornece um conjunto de aspetos de interação (pan, zoom e click) e é também configurável para se poder por exemplo usar um mapa de fundo com imagens de satélite ou de estradas conforme o que necessitarmos.

As vantagens desta versão em relação à versão cliente-servidor é não necessitar de fazer pedido ao servidor quando se dá um evento que apenas altere a camada temática (alteração na legenda ou do controlo temporal), nem haver necessidade de existir um servidor que trate da camada temática pois a mesma é tratada do lado do cliente.

Nos dias que correm existem várias tecnologias que podem ser usadas para a construção de mapas temáticos na web, tornando interessante o uso da arquitetura standalone.

Existem três tecnologias que são usadas para esta arquitetura, as quais veremos de seguida, sendo todas APIs para construção de gráficos.

Scalable Vector Graphics(SVG) é uma tecnologia com formato XML, para descrever de forma vetorial desenhos e gráficos a duas dimensões quer de forma estática, quer de forma dinâmica. Uma das principais caraterísticas dos gráficos vetoriais é de não perderem qualidade ao serem ampliados. Uma das grandes vantagens do uso de SVG é ser possível associar eventos a objetos da figura SVG, isto porque os elementos estão na DOM(Document Object Model) e como tal é possível associar eventos como a qualquer outro elemento da DOM.

O SVG permite, no contexto de criação de mapas temáticos ou qualquer outro tipo de mapa, oferecer uma grande facilidade na execução de operações interativas. Com o SVG é possível que a arquitetura standalone funcione bem, sendo o SVG simplesmente incorporado numa página HTML e os elementos gerados através da linguagem JavaScript. Numa arquitetura standalone e usando SVG o servidor irá apenas enviar os dados iniciais do mapa, ficando o cliente com a tarefa de gerar o mapa a partir dos dados que o servidor facultou. Quando o número de elementos a serem mostrados é muito grande, então existirão um grande número de elementos na DOM o que leva à necessidade do uso de bastante memória. Em mapas com milhões de pontos vai-se tornar num entrave a uma interação fluída.

Canvas, ao contrário de SVG, apenas utiliza um elemento na DOM, independentemente do número de elementos a desenhar. Como só é desenhado um elemento na DOM, então mesmo para uma quantidade elevada de objetos a desenhar, a memória consumida vai ser inferior em relação ao SVG. Usando o elemento Canvas toda a manipulação é feita através de JavaScript usando o contexto associado e as funções de desenho disponíveis.

O elemento Canvas pode ser visto com uma tela que está presente na DOM e é preenchida através de JavaScript, sendo o resultado final equivalente a uma imagem. Canvas manipula os pixeis diretamente, ou seja, desenha pixeis e nada mais. Logo, não existe maneira, usando Canvas, de alterar o que foi desenhado ou reagir a eventos. Se quisermos manipular a imagem no Canvas temos de a redesenhar. Tal como quando usamos SVG, é possível com Canvas usar a arquitetura standalone, não necessitando do servidor para gerar os mapas temáticos, sendo os mesmos gerados pelo cliente com base nos dados que o servidor enviou.

WebGL é baseado em OpenGL e corre nos browsers através do elemento Canvas, depois de ir buscar o contexto ao elemento referido. Todos os pedidos são feitos através de JavaScript. O WebGL usa a GPU(Graphics Processor Unit) permitindo assim melhoria na velocidade de renderização dos objetos. Quando usamos WebGL tendo em conta os benefícios que esta tecnologia apresenta, permite-nos trabalhar com grandes quantidades de objetos de uma forma fluída, permitindo uma melhor interação com os mapas temáticos.

Tal como quando se usa SVG ou o Canvas, com WebGL é possível tirar partido da arquitetura standalone, sendo o WebGL usado para gerar os mapas temáticos do lado do cliente através dos dados fornecidos pelo servidor. O WebGL foi usado pelo Diogo Cardoso na tese que precedeu esta e na qual foi desenvolvida a API Gisplay que irá ser alvo de escrutínio na próxima subsecção. A API Gisplay é um bom exemplo do uso da arquitetura standalone para a criação de mapas temáticos.

2.3.2 APIs Web para Mapas Temáticos

Como o objetivo é adicionar a componente temporal a mapas temáticos num contexto de uma API é importante perceber que APIs existem e o que estas fazem para conseguir mapear a componente temporal.

2.3.2.1 Google Maps

A API Google Maps [25] é possivelmente a API de mapas mais popular, devido à enorme popularidade do Google Maps e Google Earth, sendo estes serviços utilizados praticamente em todo o mundo. A API Google Maps surgiu em 2005, sendo uma API grátis mas com algumas restrições, nomeadamente ao nível da geração de mapas por dia, que é de 25000 por dia e o tamanho dos ficheiros de dados está limitado em termos de tamanho máximo permitido. Embora a API Google Maps não tenha sido desenvolvida a pensar na criação de mapas temáticos, esta pode ser usada para este efeito, pois permite aos programadores criarem os seus próprios mapas e adicionar elementos como pontos, linhas e polígonos, utilizando a tecnologia Canvas.

Em termos positivos esta API é fácil de usar, bastando ter um ficheiro de dados num dos formatos suportados. Por outro lado, pela negativa, (1) impõe um limite na utilização diária e no tamanho dos ficheiros de dados que é possível carregar, (2) não foi feita a pensar em mapas temáticos, (3) o tempo de carregamento dos ficheiros é demasiado

longo e (4) utiliza o elemento Canvas do HTLM5 que tem uma performance inferior ao WebGL quando se está a lidar com grandes quantidades de dados.

2.3.2.2 Leaflet

Leaflet é uma API criada por Vladimir Agafonking [2]. Esta API é relativamente pequena em tamanho ocupado, sendo que a última versão (1.2.0) lançada em Agosto de 2017 tem apenas 38KB (*gzipped*) de código JavaScript.

Leaflet tem como objetivo ser de fácil utilização por parte do programador, contando com uma vasta biblioteca de funções para interagir com mapas (e.g., adição de marcadores, polígonos ou linhas). Como anteriormente referido o Leaflet é relativamente leve do ponto de vista de espaço digital ocupado, sendo que consegue obter isto porque a biblioteca core contém as funções essenciais e qualquer funcionalidade adicional é estendida através de plugins. Em termos de plugins, são suportados vários fornecedores de mapas de fundo, tal como o Google Maps ou Bing Maps, o que permite variedade na criação dos mapas. Os formatos dos dados suportados são GeoJSON, TopoJSON, CSV e KML, ou seja, os formatos mais conhecidos estão todos disponíveis. A API é bastante extensível pois conta com uma comunidade bastante extensa que desenvolve plugins para as mais variadas necessidades. Um exemplo que usa a tecnologia WebGL é o WebGL Heat Map plugin que tem como objetivo criar Heat Maps tirando partido da tecnologia WebGL para permitir um desempenho satisfatório.

Por fim, é ainda de referir que existem alguns plugins no Leaflet para adicionar a componente temporal aos mapas temáticos, tais como [37].

2.3.2.3 MapMap

A API MapMap.js é uma API desenvolvida para a criação de mapas temáticos interativos de forma rápida na web [38]. Em relação à representação e construção gráfica dos mapas, é utilizada a tecnologia SVG para que os mesmos sejam atrativos para o utilizador.

Como é usado a tecnologia SVG, poderão surgir problemas de desempenho quando o conjunto de dados a representar for muito grande, isto porque, o SVG não é escalável. Em termos de formatos de dados aceita GeoJSON, TopoJSON e CSV. A API pretende oferecer uma ferramenta poderosa para a construção de mapas temáticos, no entanto sofre de problemas de performance com grandes volumes de dados pois como foi referido utiliza a tecnologia SVG.

2.3.2.4 CesiumJS

A API CesiumJS[2], é uma API para criação de visualizações a três dimensões do globo e visualização de mapas, usando a tecnologia WebGL. Não é uma API diretamente criada para criação e visualização de mapas temáticos, mas tem a possibilidade de desenhar

[2] https://cesiumjs.org

linhas, pontos e polígonos. Para criar, por exemplo, um mapa do tipo Choropleth, esta API requer que o programador escreva muito código porque não disponibiliza um nível de abstração adequado necessária para a criação de mapas temáticos. Outro exemplo da falta de abstração para a criação de mapas temáticos é a criação da legenda do mapa. Esta precisa de ser completamente escrita pelo programador sem quaisquer automatismos.

2.3.2.5 OpenLayers

OpenLayers[3] é uma biblioteca de código aberto que permite criar mapas temáticos. Não sendo uma API dedicada à construção de mapas temáticos, permite construí-los usando a tecnologia SVG e Canvas. Hoje em dia, já permite o uso da tecnologia WebGL para a criação das primitivas geométricas mas é ainda bastante limitado pois ainda está em desenvolvimento. Como tal, ainda não tem a performance que se deseja para uma API de mapas temáticos client-side.

2.3.3 Gisplay

A API Gisplay é uma API que se dedica à construção de mapas temáticos na web, tirando partido das caraterísticas intrínsecas da tecnologia WebGL.

O principal objetivo da API Gisplay é dar ao programador uma forma rápida e acessível para criar mapas temáticos, precisando apenas de um objeto de dados (por exemplo GeoJSON), um mapa de fundo e um conjunto de opções de personalização do mapa. É ainda caraterística desta API a manutenção de um desempenho aceitável mesmo quando está a lidar com mais de um milhão de pontos, permitindo mesmo neste caso extremo uma interação fluída.

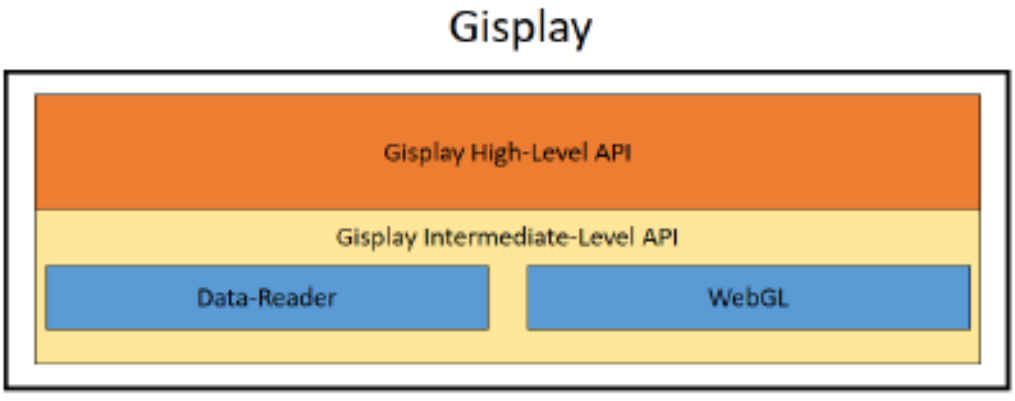


Figura 2.36: Diagrama da arquitetura da API Gisplay [17].

Gisplay, como é possível ver na figura 2.36, é composta por duas APIs uma denominada Gisplay High-Level API e a outra denominada Gisplay Intermediate-Level API, que

[3] https://openlayers.org

são, respetivamente, a API de alto nível e API de nível intermédio. Quer isto dizer que a API de nível intermédio contém funções genéricas para a construção de mapas temáticos e oferece abstração para o programador no uso de WebGL, para leitura e processamento de dados. A API de alto nível contém as funções destinadas à construção dos vários tipos de mapas temáticos presentes na demonstração de capacidades da API (Choropleth, Change Map, Mapa de Pontos e Mapa de Símbolos Proporcionais) bem como as respetivas configurações por defeito. A API de alto nível pode ser facilmente extensível pelo programador pois foi construída a partir da API de nível intermédio.

Em termos da componente temporal a mesma não é parte integrante da API Gisplay pois não foi esse o foco da API, tendo-se focado na construção de mapas temáticos com componente espacial. O que a API faculta mais próximo da componente temporal são os mapas temáticos do tipo Change Map os quais permitem ver a evolução temporal de determinado atributo para cada área geográfica.

A API Gisplay tem ainda outros problemas que serão alvo de tratamento nesta dissertação: (1) o único formato de dados suportado é GeoJSON, (2) o esquema de cores usado não é um standard reconhecido, (3) não suporta mapas com várias projeções, (4) apenas disponibiliza duas variáveis visuais. O primeiro é um problema porque existem muitos *datasets* não estão neste formato e para se usar este formato os dados teriam de ser alvo de uma conversão. O segundo ponto é outro problema devido à necessidade de usar um sistema de cores que seja reconhecido pelos utilizadores. O ponto três é importante resolver porque caso seja preciso projetar Portugal Continental e as ilhas, precisa-mos de diferentes projeções para levar a cabo esta operação. O último problema identificado limita bastante os tipos de mapas temáticos que se podem construir com a API Gisplay.

Resumidamente, as principais **limitações** da API Gisplay são:

- Número reduzido de mapas temáticos disponíveis.
- Existência de apenas duas variáveis visuais (cor e tamanho).
- Uso da cor incorretamente implementado e tamanho apenas funcional para círculos.
- Mapas temáticos não possibilitam o uso de mais do que uma variável visual em simultâneo.
- Apenas um mapa de fundo disponível (Mapbox).
- Não existe possibilidade de ter mais do que uma projeção sobre os dados (múltiplas projeções).
- Legendas pouco adequadas em termos visuais e funcionais.
- Suporte apenas para dados no formato GeoJSON.
- Uso de bibliotecas externas para implementação da interatividade do mapa, sendo que estas bibliotecas têm um custo computacional demasiado elevado na sua criação.

- Existia uma conversão inicial de coordenadas geográficas (longitude, latitude) para coordenadas Canvas do lado do processador que afetava negativamente a performance inicial.

A API Gisplay mesmo possuindo as limitações anteriormente identificadas, particularmente em relação à componente temporal, possui por outro lado várias caraterísticas, tais como o uso de WebGL e divisão em API de alto nível e API de nível intermédio, que a tornam uma boa candidata para uma possível iteração, sendo esta iteração explicada no próximo capítulo onde serão apresentados os objetivos para a API Temporal Gisplay.

2.4 Tecnologia WebGL

A tecnologia WebGL permite trazer para o browser as capacidades gráficas das aplicações nativas e desta forma reduzir a discrepância nas capacidades gráficas entre os dois tipos de aplicações. A API Gisplay optou por utilizar esta tecnologia porque a mesma permite uma performance sem paralelo em relação às alternativas existentes no browser, algo que será mantido na API Temporal Gisplay.

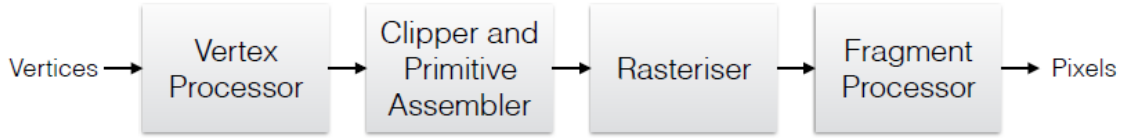


Figura 2.37: Pipeline da tecnologia WebGL

Hoje em dia as placas gráficas usam o chamado *Programmable Pipeline* que permite que os programadores consigam escrever pedaços de software especiais, chamados shaders, que descrevem o comportamento de parte do pipeline. O *WebGL* é baseado no *OpenGL ES* e tal como este utiliza a linguagem GLSL (*OpenGL Shading Language*) para escrever os programas que serão executados pela placa gráfica.

Na figura 2.37 é possível ver um diagrama que exemplifica a forma de funcionamento do *pipeline WebGL*, em que do lado esquerdo entram vértices e do lado direito temos os pixeis resultantes.

O *WebGL* tem disponível apenas as primitivas: ponto, linha e triângulo, sendo que a partir do triângulo se cria qualquer outra figura geométrica. Para desenhar as primitivas estas têm de ser enviados para a placa gráfica em grupos de vértices juntamente com os dados associados. Sendo assim, desenham-se primitivas através do agrupamento de um ou mais vértices, em que para os triângulos se fazem grupos de três vértices enquanto que para as linhas os agrupamentos são feitos dois a dois.

Existem dois tipos de *shaders* suportados na arquitectura *WebGL*, *vertex shader* e *fragment shader*. No pipeline da figura 2.37 é possível ver o *vertex processor* que é onde corre o *vertex shader* e o *fragment processor* que é onde corre o *fragment shader*.

Para que o processo de desenho se inicie é necessário que o programador faça uso de uma das funções existentes para desenhar primitivas, sendo que o *WebGL* disponibiliza dois métodos para este desenho: ***drawArrays()*** e ***drawElements()***.

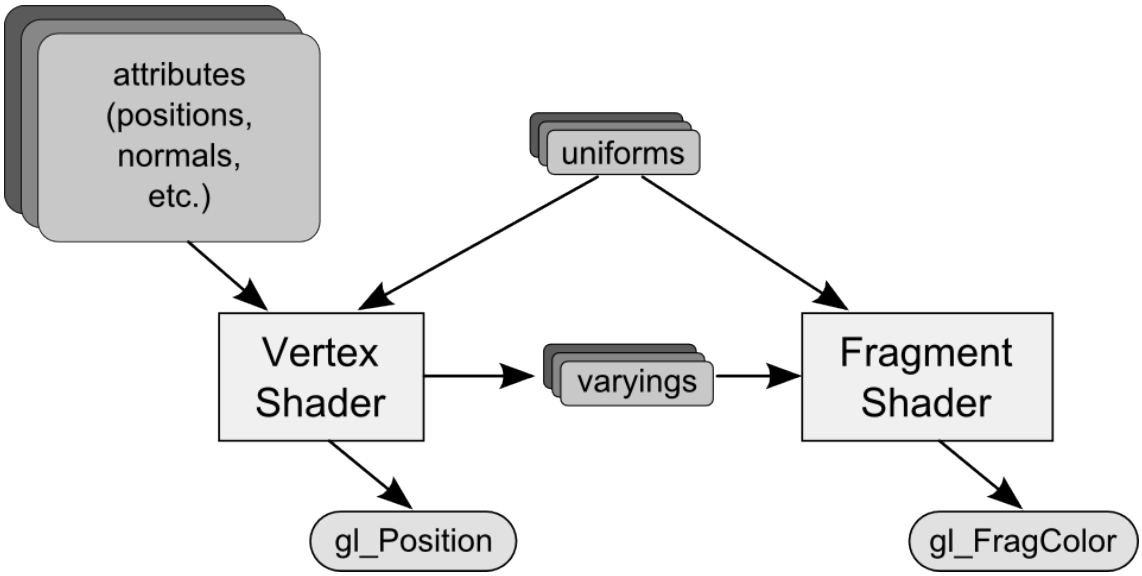


Figura 2.38: Fluxo de informação no *WebGL*.

A figura 2.38[4] mostra o fluxo de informação no *WebGL*. Como foi dito anteriormente são enviados vértices através do seu agrupamento dependendo do tipo de primitiva que se pretenda desenhar. Os vértices são transformados no *vertex shader* e o resultado desta transformação é uma nova posição (*gl_Position*). Depois, esta nova posição juntamente com a informação relativa à primitiva (valores *uniform*) que se está a desenhar vai ser usada para preencher os pixeis que estarão no seu interior (e.g. os pixeis que estão no interior de um triângulo ou os pixeis que compõe uma linha). O *fragment shader* tem como principal objetivo decidir qual a cor de cada um destes pixeis, estando no final o seu resultado na variável *gl_FragColor*. Da figura anterior fica a faltar os *varyings* que são usados para passar informação entre o *vertex shader* e o *fragment shader*.

O *WebGL* sofre de duas grandes limitações que resultam da forma como funciona. A primeira é que os valores que sejam *uniform* só se podem mudar entre duas chamadas, sendo assim os *uniform* servem para passar um valor que é global a todos os objetos de determinada chamada, sendo que por exemplo a cor ou o tamanho associado a todos os objetos a desenhar poderão ser candidatos a usar *uniforms*. A segunda principal limitação é que se quisermos ter informação (e.g. cores) a mudar durante uma única chamada só podemos associar esta informação aos vértices. Ou seja, no *WebGL* só temos dados globais para todos os vértices de uma determinada chamada ou dados associados a cada vértice.

Daqui resulta que se quisermos desenhar várias primitivas da mesma cor, isto é, por exemplo desenhar vários triângulos da mesma cor, vamos ter duas formas de proceder. A

[4] http://data.webglearth.com/doc/webgl-earthch1.html

primeira possibilidade é repetir a cor por vértice, isto porque com a interpolação que irá existir entre os vértices para obter a cor final, ao repetirmos a cor para todos os vértices esta será a cor resultante. A segunda possibilidade é agrupar todos os triângulos de uma determinada cor e depois fazer o desenho destes numa só chamada. O segundo caso é mais interessante porque permite evitar o uso de memória extra e permite fazer a divisão dos objetos por classes, sendo este o problema que temos nos mapas temáticos em que podemos dividir objetos por classes que partilhem uma ou mais variáveis visuais.

Para mapas que usem pontos este problema não se coloca porque podemos facilmente associar atributos diferentes de ponto para ponto, lembrando que como referimos anteriormente se pode associar informação a cada vértice. No entanto, para as linhas e para os triângulos isto já não se verifica porque não podemos associar diretamente informação à primitiva mas sim a cada vértice.

Concluindo, para esta dissertação vamos ter de arrumar as linhas, triângulos e pontos em grupos que partilhem os mesmos valores para as variáveis visuais para que seja possível assim desenhar de uma só vez todos aqueles que partilhem informação, diminuindo a quantidade de memória necessária para guardar a informação temática e tirando o máximo partido da tecnologia *WebGL*.

Capítulo 3

Objetivos

Neste capítulo, serão apresentados os principais objetivos a concretizar na API Temporal Gisplay, bem como a sua razão de ser. Como veremos, a API Gisplay tem várias lacunas, as quais requerem uma resolução apropriada.

3.1 Variáveis Visuais

Primeiramente, visto que as variáveis visuais são o meio usado para a transmissão da informação exposta no mapa temático para o utilizador, é essencial que as estas sejam usadas de forma correta, tanto individualmente como em grupo.

Na API Gisplay existiam apenas duas variáveis visuais disponíveis: a cor e o tamanho, além da posição que está associada à localização geográfica do elemento. Ainda assim, existindo apenas duas variáveis visuais, estas sofriam de alguns problemas, os quais são de extrema importância resolver, para como foi dito anteriormente, transmitir de forma correta a informação exposta no mapa temático para o utilizador. Estes problemas serão prontamente discutidos e serão apresentadas as soluções possíveis.

Começando pela cor, era usada a API chromajs[1], sendo esta uma API para facilitar a criação de escalas de cores e fazer conversão de cores, não sendo portanto concebida diretamente para o fornecimento de cores para mapas temáticos. No entanto, esta API possui forma de aceder aos esquemas de cores do ColorBrewer[30] e fornece forma de cálculo de classes para as variáveis contínuas. Ainda assim, esta funcionalidade (ColorBrewer) não era utilizada pela API Gisplay, o que levava a que em muitos casos os mapas usassem cores pouco adequadas aos fenómenos que pretendiam transmitir.

No caso do tamanho, este só era possível utilizar para uma forma específica, o círculo. Qualquer outra forma que o utilizador pretendesse utilizar necessitava de se modificar

[1] https://github.com/gka/chroma.js/

o *vertex shader* do código WebGL para que fosse possível desenhar essa nova forma (e.g. triângulo ou quadrado).

Como vimos, as variáveis visuais que existiam na API Gisplay possuíam alguns problemas na sua utilização, mas também não existia forma de utilizar mais do que uma variável visual em simultâneo num mapa temático. Existindo apenas uma variável visual por cada mapa temático, tende a limitar o âmbito de utilização dos mapas temáticos e por conseguinte da própria API Gisplay. Outro problema é que por vezes, o tipo de informação que se quer representar não é possível, ou fica fortemente limitada, com apenas uma variável visual.

Sendo assim, foi proposta a adição de mais variáveis visuais, a forma, a textura e a orientação, as quais se juntam à cor, ao tamanho e à posição. Das sete variáveis visuais propostas por Bertin[12] só a saturação não fará parte das disponibilizadas pela API Temporal Gisplay. Não só, a API Temporal Gisplay fica com uma variedade de variáveis visuais bastante mais significativa do que a API Gisplay, como também é possível construir mapas em que exista simultaneidade de variáveis visuais (e.g. nos mapas de pontos usar a cor para uma variável contínua e a forma para uma variável categórica), bem como outros tipos de mapas que não seriam possíveis anteriormente, os quais serão apresentados mais à frente.

3.2 Tempo

A componente temporal não era parte integrante da API Gisplay. Neste caso, ao contrário do que acontecia com as variáveis visuais, esta lacuna deve-se à limitação de âmbito feito na dissertação anterior. A componente temporal é o estandarte desta dissertação, e como tal é necessário perceber quais os benefícios que a sua inclusão acarretam bem como possíveis mudanças necessárias na API Temporal Gisplay.

Começando pelos dados, é sempre a partir destes que se consegue criar visualizações, logo para que exista a visualização da componente temporal os referidos dados terão, como é óbvio, de incluir a componente temporal. Os dados com componente temporal podem ser divididos em duas categorias principais: dados sem entidade e dados com entidade. Neste contexto, existência de entidade significa que existem múltiplos registos associados a um individuo ou coisa, sendo que, tipicamente cada registo está associado a um momento temporal dessa mesma entidade.

Os dados sem entidade, dados que não estão ligados a um individuo ou coisa concreta, podem-se dividir em dados com polígonos ou dados com pontos. No caso dos dados sem entidade por pontos, estes são eventos espácio-temporais que se desvendam através de pontos existindo como exemplo os crimes ou acidentes de viação. Nos dados sem entidade que possuem polígonos temos, por exemplo, os fogos florestais ou os furacões. Existe ainda o caso em que os polígonos têm associados um momento inicial e um momento final do evento.

Sendo assim podemos designar cada um destes casos por:

- Pontos - (*Pt*, *t*, *Ai*) em que *Pt* é a localização geográfica do ponto, *t* é o momento temporal e *Ai* são os atributos associados.

- Polígono - (*Pl*, *t*, *Ai*) em que *Pl* é o polígono, *t* é o momento temporal e *Ai* são os atributos associados. Pode existir ainda o caso (*Pl*, *t1*, *t2*, *Ai*), que é similar ao anterior mas no qual existem dois registos temporais que marcam o tempo inicial (*t1*) e final (*t2*).

Por outro lado, temos os dados que incluem entidades, que tal como já foi referido, acontece quando existem múltiplos registos associados a um mesmo individuo ou coisa, estando tipicamente cada registo associado a um momento temporal diferente. Dentro dos dados que incluem entidades temos dois grupos distintos, no primeiro grupo a geometria varia ao longo do tempo e no segundo não existe variação da geometria ao longo do tempo.

Casos em que a geometria varia ao longo do tempo são chamados de trajetórias, existindo o exemplo das migrações humanas ou de outras espécies. As trajetórias não estão dentro do âmbito atual da API Temporal Gisplay, então por este motivo não serão incluídas nesta discussão.

No caso em que a geometria não varia ao longo do tempo dá-se o nome de séries temporais georreferenciadas. As séries temporais georreferenciadas podem ter como primitiva geométrica, pontos, polígonos ou linhas. No caso dos pontos, temos o exemplo da evolução temporal da poluição causada por uma determinada fábrica de extração de óleo. No caso dos polígonos, temos o exemplo da evolução administrativa dos países bálticos nos últimos 50 anos ou de um modo geral a evolução temporal de indicadores associados a regiões administrativas.

Por fim, para as linhas, temos o exemplo da evolução temporal da informação associada a uma estrada ou via férrea.

Sendo assim, podemos designar os três casos acima dispostos por:

- Pontos - (*Pt*, {(ti, A1...An), i=1...n}) em que cada ponto possui vários momentos temporais e em cada um desses momentos temporais existem os atributos associados.

- Polígono - (*Pl*, {(ti, A1...An), i=1...n}) em que cada polígono possui vários momentos temporais e em cada um desses momentos temporais existem os atributos associados.

- Linha - (*L*, {(ti, A1...An), i=1...n}) em que cada linha possui vários momentos temporais e em cada um desses momentos temporais existem os atributos associados.

Como foi visto na secção 2.2.6, existem duas formas principais de mapear o tempo em mapas temáticos: mapeamento estático e mapeamento dinâmico. No mapeamento estático, como o nome indica a visualização não muda ao longo do tempo, logo podemos mostrar um momento temporal ou um intervalo. Por outro lado, no mapeamento dinâmico a visualização muda ao longo do tempo sendo normalmente levada a cabo com recurso a slide-shows ou animações. Embora seja de notar que a presença ou ausência

destes mecanismos de interação para navegar no tempo não tem influência na categorização da visualização como estática ou dinâmica [5]. A API Temporal Gisplay irá contar com os dois tipos de mapeamentos referidos anteriormente e com isto permitir que sejam produzidas visualizações para um ou mais instantes, dependendo do tipo de mapa que se pretende criar.

3.3 Novos Mapas

Resultante das secções anteriores, agora é possível, com a componente temporal e com um aumento significativo das variáveis visuais disponíveis, criar novos tipos de mapas que não seriam possíveis anteriormente e também é possível estender as capacidades dos mapas que existiam na versão anterior.

Na API Gisplay, existiam quatro mapas: mapa de pontos, mapa de símbolos proporcionais, mapa do tipo Choropleth e Change Map. Na nova versão estes quatro mapas estarão presentes, pois os mesmos são bastante utilizados a nível analítico, e serão adicionados os mapas de figuras, de polígonos e de linhas.

O mapa de figuras utiliza uma figura para representar o fenómeno que tem lugar numa localização específica do mapa e tem como primitiva o ponto.

O mapa de polígonos é todo o mapa que utilize polígonos (áreas), sendo que na API Temporal Gisplay passará a ser possível utilizar a textura como variável visual para os mapas de polígonos.

Por fim, os mapas de linhas são mapas novos na API Temporal Gisplay, pois na API Gisplay só havia mapas que usavam pontos ou polígonos como primitiva geométrica. Os mapas de linhas permitem mostrar fenómenos que se liguem por dois ou mais pontos na superfície terrestre, sendo exemplo os mapas de fluxos (similar ao mapa que está a ser mostrado na imagem 2.31).

Como é referido na subsecção 2.2.5, usando mais do que uma variável visual é possível criar mapas *bivariate* (duas variáveis visuais) ou *multivariate* (mais do que duas variáveis visuais). Casos que usem duas variáveis visuais são bastante usuais permitindo relacionar vários atributos de um determinado fenómeno. Por exemplo, num mapa de pontos ter uma variável categórica a ser mapeada pela forma e outra variável contínua a ser mapeada pela cor, sendo que este exemplo não era possível de ser produzido anteriormente mas agora passará a ser possível. Casos que usem mais do que duas variáveis visuais (*multivariate*) são raros porque nestes casos aumenta significativamente a capacidade necessária por parte do utilizador para fazer a interpretação do mesmo. No entanto, a API Temporal Gisplay permitirá, se o programador assim pretender, criar mapas temáticos com mais do que duas variáveis visuais.

A API Temporal Gisplay terá de conseguir, na nova iteração, cumprir os seguintes objetivos:

- **Fornecer um nível alto de abstração** e portanto o uso da API Temporal Gisplay deve ser conciso e declarativo.
- **Fornecer os melhores valores por omissão**, em termos de cartografia e visualização de dados.
- **Alto nível de personalização**, permitindo a sobreposição dos referidos valores por omissão. Nesta medida a API deve fornecer cores por omissão e o método de cálculo de classes. No entanto, se o programador pretender pode utilizar outro método de cálculo de classes ou inclusive fornecer o seu próprio método.
- **Vários tipos de mapas temáticos**, que permitam utilizar as três primitivas geométricas (ponto, linha e polígono).
- **Ser facilmente extensível**, fornecendo mecanismos simples para criar uma variante de um dos mapas existentes ou produzir um novo de raiz.
- **Lidar com milhões de pontos**, para lidar com conjuntos de dados do mundo real.
- **Fornecer interatividade necessária para facilitar análise**, nomeadamente para seleção, zoom e pan.
- **Múltiplas projeções**, isto é, dar a possibilidade de ter várias projeções sobre os dados. Sendo particularmente importante para regiões que estejam afastadas fisicamente. É o caso de Portugal e as regiões autónomas da Madeira e Açores.
- **Uso dos fornecedores de mapas de fundo mais conhecidos**, dando liberdade de escolha ao programador e criando mecanismos que permitam adicionar novos fornecedores.
- **Facilidade de uso da legenda**, sendo efetiva em termos visuais e analíticos e fornecendo mecanismos para seleção e filtrar os dados visualizados.
- **Permitir o uso de mais do que um mapa temático na mesma vista.** Para fins analíticos ou para melhor comunicar ideias é conveniente que seja possível visualizar vários mapas temáticos da mesma região, onde cada um deles mostra uma perspetiva diferente do fenómeno, seja anos diferentes ou atributos diferentes.
- **Capacidade de utilização da componente temporal**, de forma a possibilitar a visualização e análise de determinado fenómeno ao longo do tempo.

Capítulo 4

Abordagem

Nesta dissertação existem vários aspetos fundamentais que é necessário tratar em sede de abordagem para levar a cabo os objetivos propostos. Os principais aspetos a ter em conta para esta dissertação são os aspetos temporais, as variáveis visuais e o uso de várias projeções.

Começando pelos aspetos temporais, estes são de grande importância e como tal têm grande impacto. Nomeadamente têm impacto nos formatos externos dos dados na medida em que os dados têm informação temporal relevante. Têm potencial impacto no volume de dados existentes, visto que potencialmente existem dados de vários instantes temporais. Têm também impacto na organização interna dos dados devido à existência de mais uma componente de indexação dos dados a ter em conta. Os aspetos temporais acabam ainda por ter impacto na interface e na interação, uma vez que na interface é necessário agora introduzir um controlador temporal e é também essencial ter forma de interagir com o mesmo. Na API Temporal Gisplay o tempo será tratado de forma discreta e pode ser cíclico ou linear. Os dados serão arrumados em unidades temporais com a resolução que a visualização pretender utilizar (e.g. hora) e a resolução mínima que é usada para organizar os dados é chamada **grânulo temporal**.

As variáveis visuais terão impacto bastante profundo porque serão introduzidas novas variáveis visuais, sendo que o impacto é ainda maior pela particularidade da possibilidade de uso combinado de múltiplas variáveis visuais num mapa temático. Esta combinação de variáveis visuais leva a um impacto na organização interna dos dados e também na interação com o *WebGL*.

As variáveis presentes nos dados são mapeadas para variáveis visuais. Os dados são organizados por **combinações de variáveis visuais** para permitir tirar partido das capacidades do *WebGL*, nomeadamente a capacidade de desenhar todos os objetos que partilham determinada caraterística (e.g. cor) de uma só vez.

Com a existência da possibilidade de gerar mapas com várias projeções introduz-se a necessidade de gestão do *layout* para combinar as várias projeções e tem impacto no desenho e redesenho do mapa temático, na legenda e na interação. Em particular quando existem várias projeções significa que existem vários *viewports*, existindo um *viewport* para cada projeção e levando a que se tenha de desenhar em cada um deles individualmente.

Naturalmente para além destes aspetos acima apresentados existem adicionalmente algumas outras componentes que foram, por necessidade ou por opção, revistas e melhoradas. Nomeadamente, o uso da cor, legendas e mapas de fundo.

Verifica-se que o desenho e implementação da API Temporal Gisplay obrigou a redefinir o quadro de referência das tarefas conceptuais e rever a arquitetura, sendo estas vistas de seguida.

4.1 Tarefas conceptuais

Para a construção de mapas temáticos existem várias tarefas conceptuais que são executadas cronologicamente. Estas tarefas são:

- **Leitura e tratamento de opções do programador**
  Nesta tarefa são lidas as opções dadas pelo programador para o mapa temático, tais como número de classes, variáveis visuais a utilizar, mapa de fundo a utilizar, entre outros. Se todas as opções estiverem dentro dos parâmetros aceites, então estas serão o mote para a maior parte das tarefas conceptuais seguintes.

- **Leitura e carregamento dos dados**
  Esta tarefa passa pela leitura dos dados, seleção dos atributos/colunas a carregar e armazenamento dos dados numa estrutura de dados interna que é independente do formato dos dados lidos.

- **Computação de classes**
  Nesta fase, são aplicados os métodos de cálculo de classes que foram especificadas pelo programador, estando como é óbvio dependente do tipo de mapa, do tipo de dados e do número de classes desejadas.

- **Criação de combinações de variáveis mapeadas diretamente no mapa temático**
  Nesta fase tendo já as classes calculadas para cada uma das variáveis contínuas e as categorias das variáveis categóricas, faz-se o produto cartesiano de todas as variáveis que são mapeadas no mapa temático e obtém-se todas as combinações possíveis destas variáveis.

- **Distribuição dos dados carregados pela respetiva combinação de variáveis**
  Tendo todas as combinações possíveis, calculadas na tarefa anterior, faz-se agora a distribuição de cada linha dos dados carregados a partir do ficheiro, para a respetiva combinação de variáveis mapeadas no mapa temático. Nesta fase, para cada uma

das combinações criadas, são lhe atribuídos os valores de cada variável visual que compõe a respetiva combinação.

- **Criação do Layout**
  Criação do Layout tendo em conta todas as opções do utilizador, tais como os limites das projeções, mapa de fundo a utilizar e número de variáveis mapeadas no mapa temático. O espaço no ecrã é dividido de forma a possibilitar a visualização da(s) vista(s), do controlo temporal e do número de legendas que será necessário criar.

- **Construção das Legendas**
  A legenda é um elemento importantíssimo na compreensão do mapa temático e na maioria dos casos é mesmo indispensável, pois sem a mesma é muito difícil o utilizador fazer uma análise acertada do fenómeno a ser mostrado. A criação da legenda é feita com base nas classes das variáveis contínuas ou nas categorias das variáveis categóricas e também está dependente do tipo de variável visual que foi escolhida para cada caso.

- **Construção do Controlo Temporal**
  O controlo do momento temporal ou intervalo de observação é uma ferramenta essencial para perceber a diferença de determinado atributo entre diversos momentos no tempo. Nesta fase é construído o controlo temporal, seja este para instantes, intervalos ou para animação.

- **Construção do mapa temático**
  Nesta tarefa é feito o desenho dos dados de acordo com a primitiva geométrica respetiva, das combinações que estão ativas na legenda e do instante/intervalo que está ativo no controlo temporal, dando assim origem ao mapa temático.

4.2 Arquitetura

Na figura 4.1, está presente uma visão geral do diagrama da arquitetura da API Temporal Gisplay. A arquitetura está claramente dividida em duas camadas lógicas, sendo que cada uma delas fornece a sua própria API que visa um objetivo diferente.

Primeiramente, a API Temporal Gisplay de alto nível foi desenhada para ser usada pelos programadores de modo a permitir a criação de mapas temáticos de forma concisa e declarativa. O módulo **Data Reader** é um módulo extensível que foi desenhado para expor diferentes leitores de dados para diferentes formatos de dados. Cada formato de dados terá de respeitar o formato final dos dados da API Temporal Gisplay.

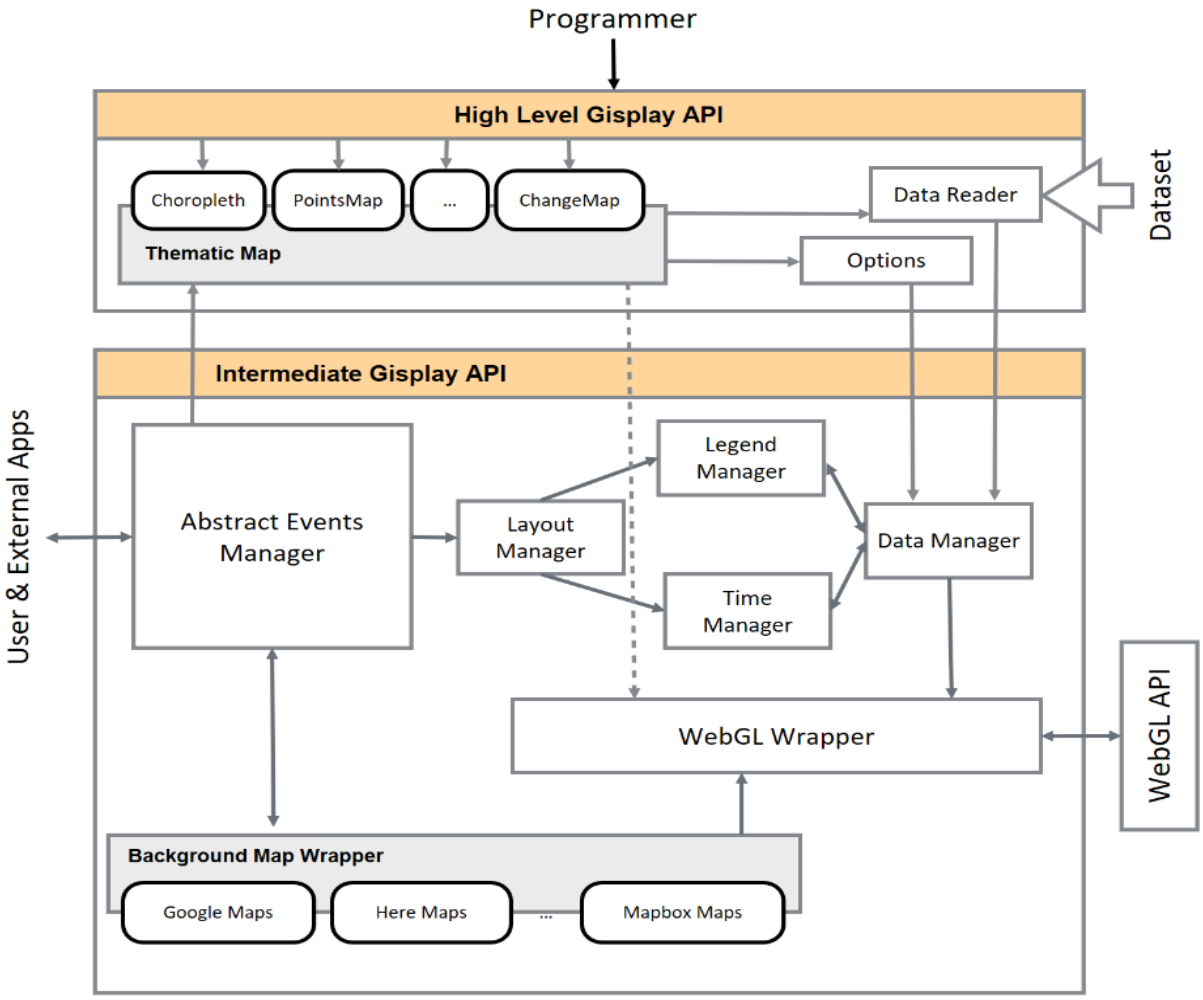


Figura 4.1: Arquitetura da API Temporal Gisplay.

O componente chamado **Options** lida com as opções especificadas pelo programador para construir mapas temáticos, desde as opções de processamento, opções de mapeamento e outras opções globais, tais como limites das vistas, fornecedor de mapa de fundo, entre outros. Para as opções que não são fornecidas pelo programador, e não sendo estas obrigatórias, a API usa o respetivo valor por omissão.

Cada um dos mapas temáticos que façam parte da API Temporal Gisplay implementam a classe abstrata base **GisplayMap**, que possui métodos gerais para a construção de mapas temáticos, tais como desenho de pontos, desenho de polígonos entre outros. Desta forma minimiza o esforço necessário para implementação de um novo mapa temático.

A API Temporal Gisplay de nível intermédio fornece métodos para construir mapas temáticos específicos através da mesma. Um componente muito importante é o **WebGL Wrapper** que fica responsável pelos aspetos relacionados com o uso da tecnologia *WebGL* neste contexto, tal como compilar *shaders*, fazer *link* dos mesmos num programa e configurando os *buffers* para serem usados pela placa gráfica. Um aspeto muito importante

do *vertex shader* é que este lida com a projeção de coordenadas geográficas para coordenadas normalizadas *WebGL*. A projeção *Web Mercator* é executada para cada vértice em paralelo pela placa gráfica, sendo esta uma vantagem muito importante do uso da tecnologia *WebGL*, evitando que o processador esteja ocupado durante milhões de ciclos a fazer operações de vírgula flutuante.

Depois, o componente **Abstract Events Manager** fica com a função de passar os eventos desde e para o mundo externo. Manipuladores de eventos externos podem ser registados na API, recebendo informação dos eventos de *zoom*, *pan* e clique que ocorrem quando há interação com o mapa temático. Na direção oposta, são providenciadas funções para serem chamadas externamente que podem ser usadas para receber informação do clique num objeto gráfico do mapa temático.

O **Legend Manager** lida com os eventos da legenda. Ao acontecer um clique numa das categorias/classes dispostas na legenda, esta irá passar a informação sobre quais os elementos atualmente ativos na mesma e desta forma serão desenhados os objetos gráficos que a estes dizem respeito.

O **Time Manager** lida com eventos relacionados com o controlo temporal. Quando se dá uma mudança dos instantes ou intervalos atualmente ativos no controlador temporal, este componente informa o mapa temático para que exista um redesenho de modo a refletir a dita alteração.

O **Background Map Wrapper** é o componente que interage com o resto da API Temporal Gisplay e esconde os detalhes de cada mapa de fundo a ser usado. Desta forma torna-se fácil a um programador adicionar novos mapas de fundo, necessitando apenas de implementar os métodos específicos desse novo mapa de fundo, tais como, os eventos de *zoom*, *pan* e clique de modo a disponibilizar o resultado deste na forma que a API Temporal Gisplay espera.

Por fim, o **Layout Manager** é o componente responsável pelo layout da API Temporal Gisplay. Este layout tem em consideração o espaço disponível no ecrã e divide esse mesmo espaço pelo mapa temático (uma ou mais vistas), pelo controlador temporal e pelas legendas existentes.

4.3 API Alto Nível

Na API de alto nível são oferecidos um conjunto de mapas temáticos, os quais podem ser criados na API Temporal Gisplay através do fornecimento de um conjunto de opções e do respetivo *dataset*.

Atualmente a API Temporal Gisplay oferece o mapa do tipo Choropleth e ChangeMap, mapa de pontos (DotMap), mapa de símbolos proporcionais (Proportional Symbols), mapa de figuras (FiguresMap) e mapa de linhas (LinesMap). Estando na tabela 6.1 cada um dos mapas anteriormente referidos e a respetiva disponibilidade de cada uma das variáveis visuais existentes na API Temporal Gisplay.

Mapa temático	Cor	Tamanho	Forma	Textura	Orientação
Choropleth	Sim	Não	Não	Sim	Não
ChangeMap	Sim	Não	Não	Sim	Não
DotMap	Sim	Não	Sim	Não	Não
ProportionalSymbols	Sim	Sim	Não	Não	Não
FiguresMap	Sim	Sim	Não	Não	Sim
LinesMap	Sim	Não	Não	Não	Não

Tabela 4.1: Mapas temáticos fornecidos na API Temporal Gisplay e respetiva disponibilidade de cada variável visual.

Começando pela cor, esta está disponível em todos os mapas, isto deve-se à cor ser uma das variáveis visuais mais importantes. Depois, o tamanho está disponível nos mapas de figuras e de símbolos proporcionais. A forma pode ser usada apenas pelo mapa de pontos, não sendo possível em nenhum dos outros. Poderia ser discutida a introdução da forma no mapa de símbolos proporcionais mas neste caso complicaria bastante a legenda e a dificuldade de compreensão do mapa temático. A textura pode ser utilizada por mapas que tenham como primitiva geométrica os polígonos, logo está disponível para o Choropleth e ChangeMap. A orientação é apenas facultada no mapa de figuras permitindo construir mapas em que as figuras possuem diferentes orientações.

Para a API de alto nível foram criados vários *wrappers* em que cada um desses *wrappers* tem a cargo a criação do seu mapa respetivo. A classe *Gisplay* tem cada uma destas funções (*wrappers*) e todas elas têm um nome idêntico começado por ***make***.

Exemplificando, para criar um mapa do tipo Choropleth existe o seguinte *wrapper*: ***makeChoropleth(parsingOptions, mappingOptions, globalOptions)***. Esta função recebe três argumentos que são as opções que o programador pode passar para a criação do referido mapa Choropleth. Na API Temporal Gisplay, as opções existentes para a criação de mapas temáticos foram divididas em três grupos. O primeiro parâmetro diz respeito às opções de processamento dos ficheiros de dados (**parsingOptions**), o segundo parâmetro diz respeito ao mapeamento das variáveis para variáveis visuais (**mappingOptions**) e o terceiro refere-se a opções globais (**globalOptions**) da API Temporal Gisplay.

Na figura 4.2 é possível ver um exemplo do *layout* da API Temporal Gisplay, no qual temos um mapa de pontos para os vários estados dos Estados Unidos da América e onde é possível ver a divisão do espaço disponível por três projeções distintas. Do lado esquerdo temos em cima o Alaska e em baixo as ilhas do Hawaii, enquanto que do lado direito temos a parte continental dos Estados Unidos. Este tipo de *layout* não era possível na API Gisplay pois nesta API só era possível ter uma projeção dos dados.

Na API Gisplay só era possível usar uma variável visual de cada vez, sendo esta uma limitação da quantidade e qualidade de mapas que se podia criar com essa API. Na API Temporal Gisplay é possível usar mais do que uma variável visual por cada visualização. Para que os valores das variáveis visuais se entendam no mapa temático temos de ter forma de saber o que cada valor significa, sendo utilizada a legenda. Na figura 4.3 é

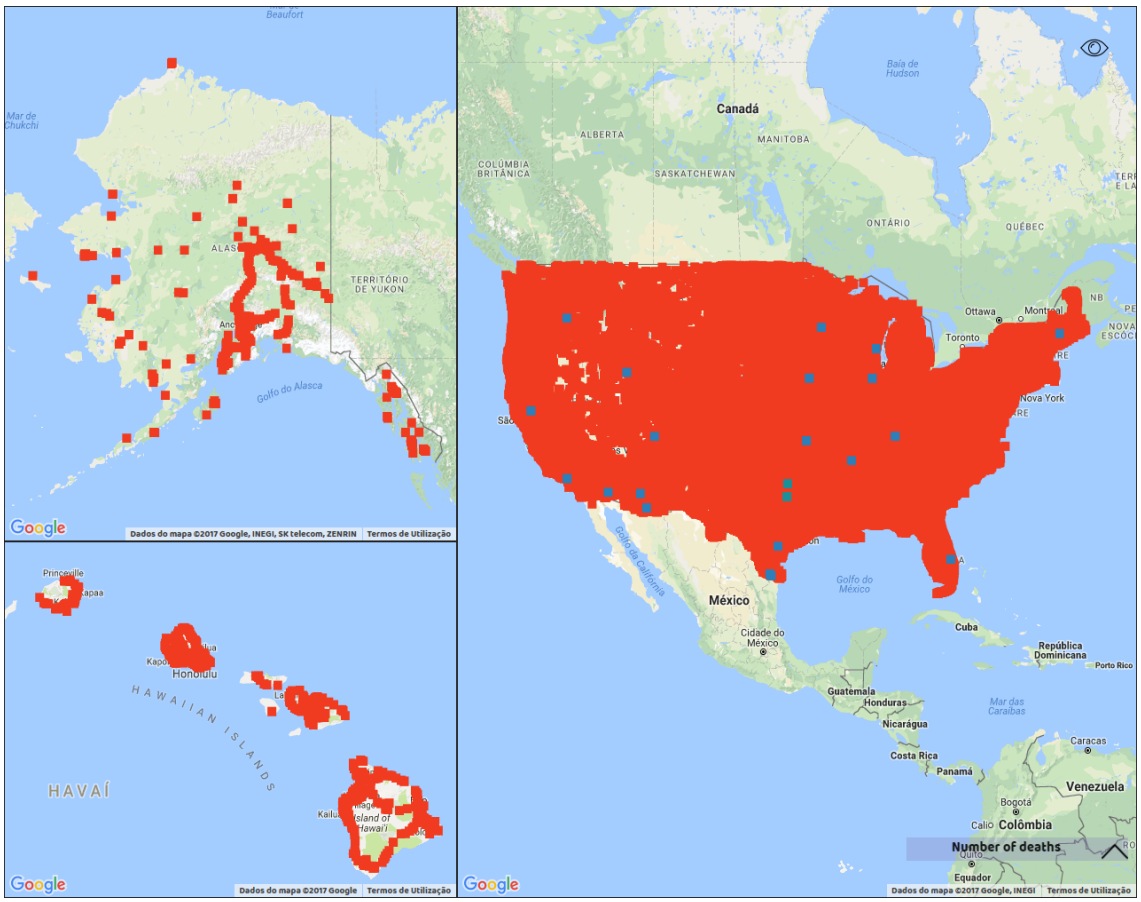


Figura 4.2: Exemplo de mapa de pontos com múltiplas projeções na API Temporal Gisplay.

possível ver um exemplo de um mapa temático que possui várias legendas. Uma legenda mostra, através da forma, o envolvimento (círculo) ou não envolvimento (quadrado) de álcool nos acidentes. A segunda legenda mostra o número de mortes através da cor. Como é possível verificar todos os pontos presentes na imagem são da cor mais clara da escala sequencial que existe para o número de mortes em cada acidente, logo isto diz-nos que todos os pontos desta imagem têm entre zero e seis mortos. Por outro lado a quantidade de pontos que são círculos e que são quadrados é quase idêntica o que nos leva a concluir que nesta zona existem aproximadamente o mesmo número de acidentes com álcool envolvido e sem álcool envolvido.

Na API Temporal Gisplay com a adição da componente temporal era necessário existir forma de controlar esta componente, independentemente da visualização ser de instantes ou intervalos temporais. Na figura 4.4 é possível ver dois exemplos em que se pode comparar dois momentos temporais num mapa Choropleth. Na figura 4.4a o controlador temporal está a selecionar o ano de 1991 e na figura 4.4b o controlador temporal está a selecionar o ano de 2007, sendo possível comparar as diferenças entre os dois.

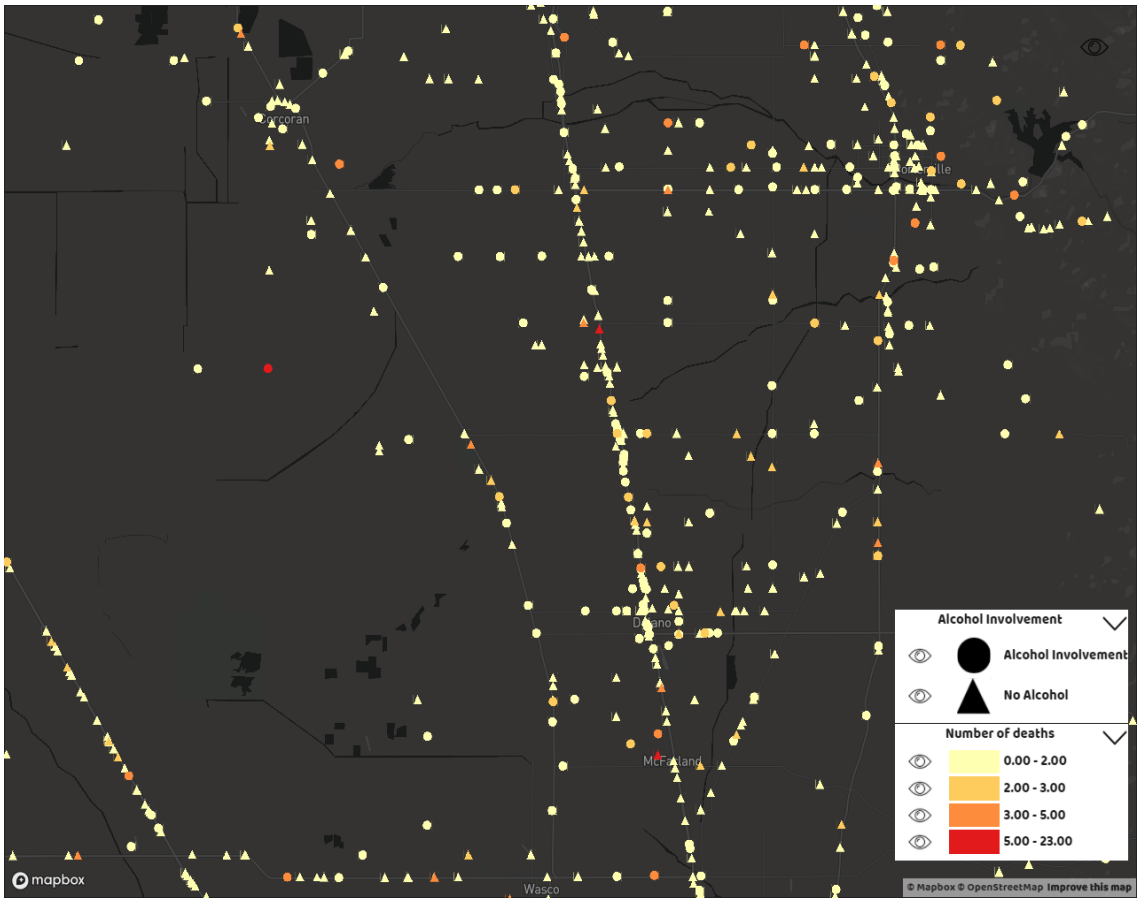
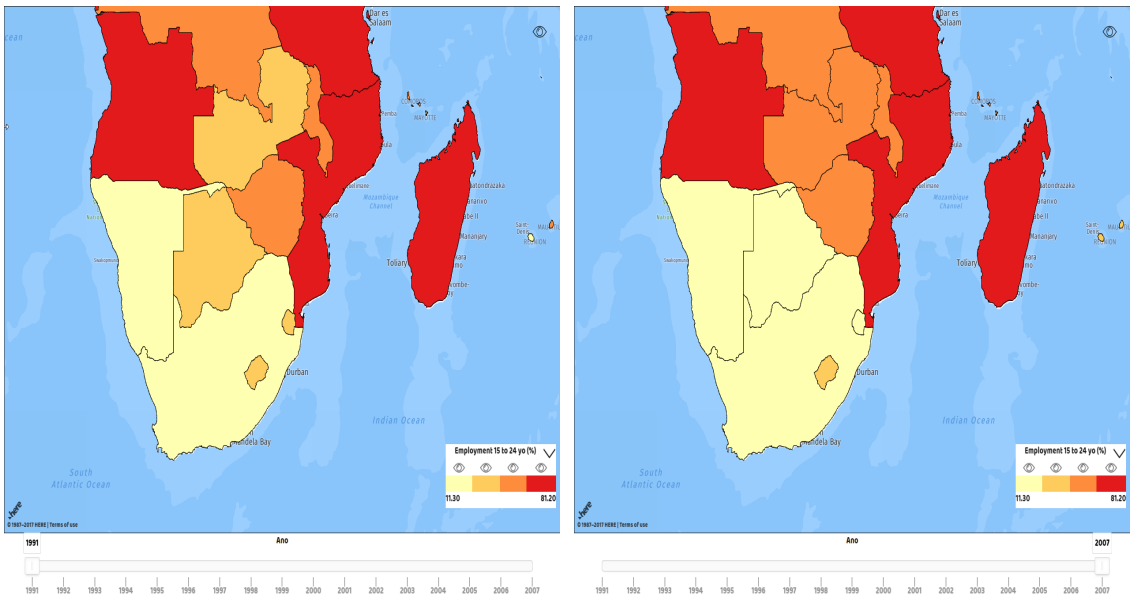


Figura 4.3: Exemplo de mapa de pontos com duas legendas na API Temporal Gisplay.



(a) Exemplo para o ano 1991.

(b) Exemplo para o ano 2007.

Figura 4.4: Exemplo de Choropleth com controlo temporal na API Temporal Gisplay.

4.4 API Nível Intermédio

A API de nível intermédio tem como principal objetivo funcionar como ponte entre a API de alto nível e a tecnologia *WebGL*, através do fornecimento de abstrações para lidar com esta tecnologia.

A classe *GisplayMap* é a classe a partir da qual todos os mapas temáticos são criados. Nesta classe estão presentes todos os métodos que são comuns a vários mapas temáticos, permitindo assim que vários mapas temáticos diferentes consigam fazer reutilização do código desses métodos.

```
drawToMap() {
    clear current drawn thematic map;

    let granulesIndices = get active temporal indices;
    let mvcs = get map variable combinations;
    let listOfBGMaps = list of background maps;
    for (let mvc of mvcs) {
        if (mvc is active) {
            let tgs = mvc.getTemporalGranules();
            for (let index of granulesIndices) {
                let tg = tgs[index];
                if (tg contains at least one  element)
                    for (let bgmap of listOfBGMaps)
                        draw(mvc, tg, bgmap);
            }
        }
    }
}
```

Listagem 4.1: Pseudo-código do método de desenho da API Temporal Gisplay

Na listagem 4.1 é possível ver o pseudo-código para o método de desenho de qualquer mapa temático da API Temporal Gisplay. Este método começa por obter a lista de índices da componente temporal que estão atualmente ativos e a lista de combinações de variáveis visuais existentes. Depois para cada combinação de variáveis visuais existentes só irá efetuar o desenho daquelas que estão ativas na legenda. Para cada uma das combinações atualmente ativas, irá percorrer todos os índices temporais ativos. Para cada índice temporal se o mesmo tiver pelo menos um elemento então será feito o desenho dos seus elementos em cada uma das projeções respetivas.

4.4.1 Desenho do mapa temático

Como os dados estão divididos pelas combinações de variáveis visuais, e dentro destas, sub-divididos em grânulos temporais, então para o seu desenho em cada projeção do mapa tem de ser chamado o método de desenho: ***draw(mvc, tg, bgmap)***). Sendo ***mvc*** a combinação de variáveis visuais, ***tg*** o grânulo temporal e ***bgmap*** o mapa de fundo no qual é para desenhar, tendo o mesmo um determinado *viewport* que é usado no *WebGL*.

```
draw(mvc, tg, bgmap) {
    this.initialDrawSetup(mvc, bgmap);

    this.createImageTexture(GisplayDefaults.SHAPE());
    this.setShapeIndex(mvc);
    this.setSizeUniform(mvc);
    this.setPoints(tg);
}
```

Listagem 4.2: Código do método draw(...) do mapa de pontos

```
draw(mvc, tg, bgmap) {
    this.initialDrawSetup(mvc, bgmap);

    this.createImageTexture(GisplayDefaults.TEXTURE());
    this.setTextureIndex(mvc);
    this.setPolygons(tg);

    this.initialDrawSetup(mvc, bgmap, true);
    this.setPolygonsBorders(tg);
}
```

Listagem 4.3: Código do método draw(...) do Choropleth

Na listagem 4.2 e 4.3 temos, respetivamente, o código do método de desenho do mapa de pontos e do Choropleth.

Ambos os métodos começam com a invocação do método ***initialDrawSetup(mvc, bgmap)***. Qualquer método de desenho deve começar por invocar este método, uma vez que este método é responsável pela configuração inicial feita antes de qualquer desenho no mapa temático. Nesta configuração inicial inclui-se a escolha do programa *WebGL* a utilizar, configuração do *viewport* do mapa de fundo, configuração da cor e da opacidade a utilizar e por fim configuração da matriz de projeção.

Após este primeiro método, os métodos a invocar irão depender de cada mapa temático. Neste caso em particular, em que temos o mapa de pontos e Choropleth, ambos fazem invocação do método ***createImageTexture(...)***, ao passo que outros mapas temáticos não necessitam de invocar este método. Isto porque, mapas temáticos que utilizem formas, padrões ou figuras irão ter de carregar a respetiva imagem para uma textura a ser usada pelo *shader* do mapa temático. Atualmente existem na API Temporal Gisplay três tipos de imagens que podem ser carregadas: imagem de padrões, formas ou figuras. No caso do mapa de pontos este invoca o método de criação da textura com o parâmetro que diz à API Temporal Gisplay para carregar a imagem com as formas, enquanto que para o Choropleth a imagem a ser carregada é que contém padrões. Para um mapa, como por exemplo, o mapa de figuras, a imagem a ser carregada será a que contém figuras.

Dando o exemplo das formas, estas estão numa imagem que está dividida numa grelha de 4 linhas por 4 colunas como se pode ver na tabela 4.2, em que cada um dos índices

da imagem possui uma forma. O círculo está no primeiro índice, a cruz no segundo e em cada um dos outros índices está a forma que o programador previamente lá tenha inserido.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Tabela 4.2: Índices de uma imagem de formas, padrões ou figuras

Depois de carregada a imagem respetiva, no mapa de pontos invoca-se o método ***setShapeIndex(mvc)*** enquanto que no Choropleth invoca-se o método ***setTextureIndex(mvc)***. O primeiro serve para dizer ao *shader WebGL* qual o índice da forma a utilizar enquanto que o segundo serve para carregar no *shader* do mapa Choropleth o índice da textura a utilizar. Noutro mapa, como é o caso do mapa de figuras, existe um método semelhante que fará o carregamento do índice da figura a utilizar.

No mapa de pontos prossegue-se agora com a invocação do método ***setSizeUniform(mvc)*** que vai carregar o valor do tamanho a utilizar por cada ponto. O tamanho no mapa de pontos é uniforme, ou seja, todos os pontos têm o mesmo tamanho visto que o tamanho nos mapas de pontos não é usado como variável visual.

A última coisa a fazer no mapa de pontos é a invocação do método ***setPoints(tg)*** que ativa o *buffer* de pontos do grânulo temporal, tendo estes pontos sido previamente carregados na GPU aquando da leitura dos dados. Na mesma linha existe o método ***setPolygons(tg)*** que permite fazer o mesmo mas neste caso para os polígonos do mapa Choropleth. No mapa de linhas existe um método semelhante ***setLines(tg)*** que faz a ativação das linhas do grânulo temporal.

No caso do Choropleth é ainda necessário fazer o desenho das fronteiras dos polígonos, sendo apenas necessário chamar o método que faz a ativação das fronteiras dos polígonos que se chama ***setPolygonsBorders(tg)***.

4.4.2 Picking

Na API Gisplay o suporte ao *click* num elemento do mapa obrigava à construção de estruturas de dados espaciais auxiliares. Estas estruturas tinham custos na sua criação muito elevados, sendo que no caso dos pontos era de aproximadamente 80% do tempo inicial. Existe uma alternativa que é muitas vezes usada em aplicações interativas e que se baseia no principio de ser possível desenhar todos os objetos de forma interativa no ecrã numa taxa relativamente elevada, fazer *picking* não demorará mais tempo do que o tempo de desenho. Esta solução passa por atribuir um identificador único a cada objeto e a fazer o desenho dos objetos (pontos, polígons) numa textura em vez de se desenhar no ecrã e a operação de *picking* resume-se a obter o identificador do elemento clicado pelo utilizador. Ou seja, obter o identificador do pixel que o utilizador clicou.

Esta solução tem a desvantagem de trabalhar na resolução do ecrã e no nível de zoom da visualização, sendo que, os objetos sobrepostos num pixel não serão todos selecionáveis, prevalecendo apenas um deles, sendo este o último a ser desenhado (ou o primeiro, dependendo da parametrização do algoritmo *z-buffer*). No entanto, não é uma desvantagem para esta dissertação, uma vez que para que um mecanismo de *picking* baseado no *click*, tenha precisão, é necessário que os objetos sejam facilmente distinguíveis no ecrã.

Capítulo 5

Implementação

Neste capítulo serão apresentados os aspetos de implementação que foram produzidos durante esta dissertação. Nomeadamente, as opções disponíveis na API, qual o tratamento dos dados, a forma de cálculo de classes, os mapas de fundo, as múltiplas projeções.

5.1 Opções API Temporal Gisplay

Para qualquer mapa temático que seja produzido com a API Temporal Gisplay, existem várias opções que terão forçosamente que ser fornecidas, tais como, as variáveis a serem mapeadas em variáveis visuais no mapa temático ou o *URL* dos dados a serem usados.

As opções foram divididas em três grupos diferentes:

1. **Opções de processamento (parsingOptions)** - são as opções para a seleção e renomeação dos dados durante a fase de processamento
2. **Opções de mapeamento (mappingOptions)** - são as opções das variáveis que são mapeadas para variáveis visuais.
3. **Opções globais (globalOptions)** - são opções gerais da API, tais como, limite das vistas ou qual o fornecedor de mapas de fundo.

5.1.1 Opções de Processamento (parsingOptions)

Estas opções serão importantes para serem usadas no processamento dos dados, permitindo renomear e selecionar as variáveis dos dados que se pretendam utilizar.

As opções de processamento estão divididas em:

1. Opções para variáveis que são mapeadas para variáveis visuais (**mapVariables**)
    a) Variáveis Categóricas

b) Variáveis Contínuas

2. Opções de variáveis opcionais (**optionalVariables**)

3. Opções relacionadas com a componente temporal (**time**)

4. Opções de urls para os dados (**urls**)

5. Opções do ficheiro CSV (**csv**)

5.1.1.1 Opções para variáveis que são mapeadas para variáveis visuais

Com estas opções é possível informar o *parser* de quais são as variáveis categóricas e contínuas a serem selecionadas e renomeadas bem como informação relativa às categorias ou classes das mesmas.

Estas opções, podem ser divididas pelo seu tipo: variáveis categóricas e variáveis contínuas.

Primeiro, para as categóricas temos apenas um caso:

```
mapVariables: [
    {
        internalName: 'Género',
        externalName: 'gender',
        values: ["M", "F"]
    }
]
```

Listagem 5.1: Fornecer opções da variável categórica

Como visto no pedaço de código acima, o nome interno (**internalName**) é o nome que o programador deu à variável e o nome externo (**externalName**) é o nome que a variável tem nos dados. Os valores (**values**) para uma variável categórica, são todos os valores possíveis que esta pode tomar (categorias). Nesta versão da API Gisplay os valores/categorias de uma variável categórica terão de ser fornecidos, mas numa versão futura, a API calculará estes valores sem haver então necessidade do seu fornecimento. Sendo assim, para as variáveis categóricas, o nome interno é opcional, o nome externo é obrigatório e os valores possíveis (categorias) são obrigatórios. O nome interno sendo opcional, se não for fornecido pelo programador, então vai tomar o mesmo valor do nome externo.

Depois para as variáveis contínuas temos três casos diferentes:

1. São fornecidos apenas os valores (mínimo e máximo).

2. São fornecidos os valores (mínimo e máximo) e qual o método para cálculo de classes e/ou o número de classes.

3. São fornecidas as classes.

```
mapVariables: [
    {
        internalName: 'Idade do paciente',
        externalName: 'age',
        values: [0, 108]
    }
]
```

Listagem 5.2: Fornecidos mínimo e máximo da variável contínua

```
mapVariables: [
    {
        internalName: 'Idade do paciente',
        externalName: 'age',
        values: [0, 108],
        classBreakMethod: 'quantiles',
        numberOfClasses: 3
    }
]
```

Listagem 5.3: Fornecidos mínimo e máximo, método de cálculo de classes e número de classes da variável contínua

```
mapVariables: [
    {
        internalName: 'Idade do paciente',
        externalName: 'age',
        classBreaks: [1, 22, 34, 60, 108]
    }
]
```

Listagem 5.4: Fornecidas as classes da variável contínua

Nos três casos possíveis para as variáveis contínuas, o nome interno e externo continuam a ser opcional e obrigatório, respetivamente. Por outro lado, poderá ser fornecido no intervalo de valores (**values**), o mínimo e o máximo que a variável contínua pode tomar (ver listagem 5.2).

Pode ainda ser fornecido qual o método para calcular as classes e/ou qual o número de classes que se pretende que a variável contínua venha a ter (ver listagem 5.3). Se estes não forem fornecidos a API usa os valores por omissão para o mapa temático em questão.

Em alternativa ao fornecimento dos valores, mínimo e máximo, do método para calcular as classes e do número de classes, pode-se fornecer diretamente as classes (**classBreaks**), como se pode ver na listagem 5.4.

Sendo assim, nas variáveis contínuas se forem fornecidos os valores, então é possível fornecer o método de cálculo de classes e o número de classes. Caso contrário, quando não forem fornecidos os valores do domínio terão de ser fornecidas as classes que a variável contínua possuirá.

No caso das variáveis que fazem parte da geolocalização (latitude e longitude) é-lhes dado um tratamento especial, umas vez que quando queremos utilizar a componente espacial nos ficheiros do tipo CSV é fundamental ter forma de as referir. Sendo assim, as palavras **Latitude** e **Longitude** são palavras reservadas para o nome interno. Na listagem 5.5 é possível ver um exemplo do fornecimento da latitude e da longitude através das suas palavras reservadas.

```
mapVariables: [
    {
        internalName: 'Latitude',
        externalName: 'latitudeParish',
    },
    {
        internalName: 'Longitude',
        externalName: 'LongitudeParish',
    }
]
```

Listagem 5.5: Fornecimento da latitude e da longitude

5.1.1.2 Opções de variáveis opcionais (optionalVariables)

Para as variáveis opcionais existe apenas a possibilidade de fornecer o seu nome interno e externo. As variáveis opcionais são apenas lidas e guardadas tal como estas se encontram no ficheiro de dados. As variáveis opcionais são todas aquelas que não são mapeadas no tempo, espaço ou outras variáveis visuais, mas que o programador pretenda que as mesmas sejam usadas como suporte para o click numa marca do mapa. O nome interno e externo, tal como para as anteriores é opcional e obrigatório, respetivamente. Na listagem 5.6 está um exemplo de uma variável opcional que será depois usada quando se dá um click num elemento do mapa temático.

```
optionalVariables: [
    {
        internalName: 'Nome da Cidade',
        externalName: 'city'
    }
]
```

Listagem 5.6: Exemplo de variável opcional

5.1.1.3 Opção relacionada com a componente temporal (time)

Para a componente temporal, terá na mesma de ser fornecido o nome externo, sendo que mais uma vez o nome interno é opcional. Adicionalmente temos a granularidade (**granularity**), que é obrigatória. O programador pode também definir qual o controlador temporal (**timeControl**) a utilizar para a visualização. Se o programador não fornecer o

tipo de controlador temporal então a API utiliza por omissão o instante (**instant**). Existem dois controlos temporais disponíveis: *instant* e *interval*.

```
time: {
    internalName: 'Ano',
    externalName: 'year',
    granularity: 'year',
    timeControl: 'instant'
}
```

Listagem 5.7: Exemplo do fornecimento da componente temporal

Nome da granularidade	Nome na API	Caraterística
Ano	year	Linear
Mês	month	Linear
Dia	day	Linear
Valor	value	Linear
Mês do ano	monthOfYear	Cíclica
Dia do ano	dayOfYear	Cíclica
Dia do mês	dayOfMonth	Cíclica
Hora do dia	hourOfDay	Cíclica
Minuto da hora	minuteOfHour	Cíclica

Tabela 5.1: Granularidades disponíveis na API Gisplay.

O tempo tem dois arranjos possíveis: linear ou cíclico. O arranjo linear corresponde à nossa perceção natural de tempo, na qual consideramos o tempo a proceder do passado para o futuro (i.e. cada valor do tempo tem um único predecessor e um único sucessor), no entanto, a ciclicidade é bastante comum em vários tipos de dados (i.e. relacionados com estações do ano ou médias mensais) [5].

Como podemos ver na tabela 5.1, existem várias granularidades disponíveis, tanto para arranjo linear como cíclico, das quais o programador pode escolher uma, sendo que esta será depois extraída da sua coluna dos dados no momento do processamento.

5.1.1.4 Opção de URLs para os dados (urls)

```
urls: {
    dataURL: 'hivCSV',
    geospatialURL: 'worldFile',
    idOnDataURL: 'geo',
    idOnGeoSpatialURL: 'ISO_A3'
}
```

Listagem 5.8: Exemplo de fornecimento dos *URLs*

Para os *URLs* tem de se fornecer sempre o *URL* dos dados (**dataURL**), sendo que o *URL* com identificadores (**geoSpatialURL**) e respetiva geometria é opcional. O *URL* com identificadores é usado quando se pretende que a geometria seja reutilizada. Caso

o *URL* com identificadores exista, então terão de ser fornecidas duas opções adicionais: **idOnDataURL**, e **idOnGeoSpatialURL**, que basicamente fazem a ligação entre a coluna de identidades que é usada no *URL* de dados e no *URL* de geometria.

5.1.1.5 Opções do ficheiro CSV (csv)

O fornecimento das opções para o ficheiro CSV nunca é obrigatório, e caso as mesmas não existam a API usa os valores por defeito.

```
1 csv: {
2     chunkSize: 1024*1024*20,
3     numWorkers: 8
4 }
```

Listagem 5.9: Exemplo de opções do ficheiro CSV

Primeiro, o **chunkSize** é o tamanho em bytes de cada leitura feita pelas *threads/workers*. Depois, o **numWorkers** é o número de *threads/workers* que vão processar cada parte do ficheiro em paralelo. Neste momento, a API já é capaz de decidir qual o carácter que delimita cada coluna e qual o carácter que delimita cada linha do ficheiro CSV. No futuro, usando o conhecimento adquirido, a API irá tomar as decisões ótimas para as opções acima referidas, dependendo do hardware da máquina em que se está a trabalhar.

5.1.2 Opções de Mapeamento (mappingOptions)

Todas as variáveis que foram referidas nas opções de processamento, com exceção das variáveis opcionais são justamente as variáveis a mapear em variáveis visuais do mapa. As opções de mapeamento que veremos de seguida servem precisamente para esse efeito.

Nestas opções, para cada variável que será mapeada por uma variável visual, tem de ser fornecida informação adicional que será depois usada para o referido mapeamento.

A API Temporal Gisplay conta com as seguintes variáveis visuais: cor, forma, textura, tamanho e orientação. Para cada uma delas é preciso determinar quais os mapeamentos possíveis, sendo possível ver precisamente isso na tabela 5.2.

Variável Visual	Variável Categórica	Variável Contínua
Cor	Sim	Sim
Forma	Sim	Não
Textura	Sim	Não
Tamanho	Não	Sim
Orientação	Sim	Sim

Tabela 5.2: Disponibilidade de cada tipo de mapeamento para cada variável visual.

Para cada uma das variáveis visuais e os seus mapeamentos, é importante saber quais serão as formas possíveis de fornecer esta informação.

5.1.2.1 Cor

Começando pela **cor**, o mapeamento da cor é opcional, e a razão para ser opcional é que a API Temporal Gisplay faz uso do ColorBrewer para decidir quais as cores indicadas em cada situação.

A cor pode ser mapeada de duas formas diferentes: através de uma variável categórica ou através de uma variável contínua. Para que uma variável categórica possa ser mapeada na cor, os seus valores de mapeamento (**mapping**) terão de ser dados na forma de *array* de cores, contendo em cada posição deste *array* o nome da cor ou o valor hexadecimal da cor. Na listagem 5.10, pode ser visto um exemplo em que as cores são dadas pelo respetivo nome, sendo que os nomes dados terão de estar na lista de nomes aceites pela API Temporal Gisplay. A lista de nomes de cores[1] aceites pela API Temporal Gisplay é a lista de cores da especificação do HTML e CSS.

```
color: {
    externalName: "gender",
    mapping: ["blue", "pink"],
}
```

Listagem 5.10: Exemplo de cor mapeada por variável categórica

Se verificarmos nas opções de processamento, para a variável que tem nome externo "gender", é possível ver que os seus valores possíveis são "M" e "F". Sendo assim, o mapeamento (*mapping*) será atribuído ao respetivo índice dos valores (*values*). Ou seja, "M"→"blue" e "F"→"pink", sendo que o mesmo se verificaria se a cor fosse dada em hexadecimal.

Por outro lado, para que uma variável contínua seja mapeada na cor, os seus valores de mapeamento terão de ser dados na forma hexadecimal. Dar as cores pelo seu nome no caso das variáveis contínuas não faz sentido pois como a variável é contínua, o que altera é a tonalidade da cor. Cada elemento dado no mapeamento da cor (**mapping**) na listagem 5.11 é atribuída à respetiva classe.

```
color: {
        externalName: "age",
        mapping: ["#ece7f2", "#a6bddb", "#2b8cbe"]
}
```

Listagem 5.11: Exemplo de cor mapeada por variável contínua

5.1.2.2 Forma

Depois da cor, temos a **forma**, sendo que esta pode ser mapeada por uma variável categórica. O seu mapeamento, ao contrário da cor, terá de ser dado na forma de mapa chave-valor. Cada chave é um dos valores (**values**) das opções de processamento e o valor

[1] https://drafts.csswg.org/css-color/#named-colors

associado à referida chave é uma forma (triângulo, quadrado, etc) que esteja disponível na API Temporal Gisplay. Um exemplo do fornecimento da forma está na listagem 5.12.

```
shape: {
    externalName: 'gender',
    mapping: {
        "M" : "triangle",
        "F" : "square"
    }
}
```

Listagem 5.12: Exemplo das opções para a forma

As formas atualmente disponíveis na API Temporal Gisplay são: triângulo, quadrado, círculo e cruz. Estas formas estão numa imagem sendo que a forma de funcionamento desta imagem e dos índices respetivos apresentados na secção 4.4 do capítulo anterior.

5.1.2.3 Textura

```
texture: {
    externalName: 'gender',
    mapping: {
        "M" : "points",
        "F" : "triangles"
    }
}
```

Listagem 5.13: Exemplo das opções para a textura

A próxima variável visual é a **textura**, que pode ser mapeada por uma variável categórica, e a maneira de fornecimento do seu mapeamento é em tudo idêntica à variável visual anterior, a forma. Para cada categoria da variável categórica é dado o nome do padrão que se pretende usar. Os padrões atualmente disponíveis na API Temporal Gisplay são: pontos, triângulos, vertical e horizontal. Estes padrões estão numa imagem que contém múltiplos padrões sendo a forma de funcionamento desta imagem e dos índices respetivos apresentados na secção 4.4 do capítulo anterior. Um exemplo do fornecimento da textura pode ser visto na listagem 5.13.

5.1.2.4 Tamanho

O **tamanho**, pode ser mapeado através de uma variável contínua. A forma de fornecer as suas opções passa por indicar o tamanho que cada classe desta variável irá possuir.

```
size: {
    internalName: 'age',
    mapping: [5, 40, 100]
}
```

Listagem 5.14: Exemplo das opções para o tamanho

Como podemos ver no pedaço de código 5.14, no mapeamento existem 3 tamanhos diferentes (5, 40, 100). Ou seja, cada classe da variável contínua terá o tamanho de acordo com este mapeamento. A primeira classe verá os seus elementos serem desenhados com tamanho 5x5 pixeis, a segunda com tamanho 40x40 pixeis e a terceira classe com tamanho 100x100 pixeis.

5.1.2.5 Orientação

Por fim, temos a **orientação**, sendo que nesta variável visual é possível usar tanto para uma variável categórica como uma variável contínua, e os valores a serem fornecidos dizem respeito aos graus da orientação.

Para uma variável categórica, tal como nas outras variáveis visuais que aceitem variáveis categóricas, é dado um objeto com chaves e valores, sendo a chave uma das categorias que existem nos dados e o respetivo valor é o número de graus, sendo possível ver um exemplo na listagem 5.15

```
orientation: {
    externalName: 'gender',
    mapping: {
        "M" : 0,
        "F" : 40
    }
}
```

Listagem 5.15: Exemplo das opções para a orientação com uma variável categórica

Para uma variável contínua, são fornecidos os graus da orientação, que todos os elementos de cada classe irão ter no mapa temático, sendo possível ver um exemplo na listagem 5.16.

```
orientation: {
    internalName: 'Number of deaths',
    mapping: [0, 40, 80]
}
```

Listagem 5.16: Exemplo das opções para a orientação

Resumindo, para as opções de mapeamento (**mappingOptions**), o nome externo é obrigatório, pois este permite referenciar qual a variável que estamos a referir nas opções de processamento, o mapeamento da variável (**mapping**) é obrigatório em todos os casos exceto quando a variável visual for a cor.

5.1.3 Opções Globais (globalOptions)

Por fim, temos as opções globais da API Gisplay.

- **bounds** - Quais os limites (Sudoeste e Nordeste) de cada vista. Neste momento é obrigatório para qualquer número de vistas. No futuro a API quando não é dada

esta opção, calcula os limites do ponto sudoeste e nordeste com base nos dados e cria uma vista com esses mesmos limites.

- **layout** - Qual o layout da vista acima referida. Se só existir uma vista, então não é necessário fornecer layout. Na secção sobre múltiplas vistas(5.6) encontra-se a explicação completa para os *bounds* e para o layout.

- **container** - Qual o identificador do elemento *div* da DOM que vai conter esta instância da API Gisplay.

- **provider** - Qual o fornecedor de mapas de fundo. Atualmente disponível: Mapbox, Google Maps, Bing Maps e Here Maps. Para usar o fornecedor basta usar o seu nome ou a respetiva sigla, sendo respetivamente **MB**, **GM**, **BM** e **HM**.

- **showLoader** - Mostrar o "loader" enquanto a API lê e processa os dados do(s) *URL(s)*.

- **mapOnClickFunction** - Função a ser chamada quando se clica num elemento do mapa. O clique num elemento do mapa retorna a linha ao qual este corresponde nos dados. Por omissão será mostrado um popup na posição do elemento com a informação relativa ao mesmo.

- **legendOnClickFunction** - Função a ser chamada quando se clica num elemento da legenda.

Na tabela 5.3 pode ser visto o resumo das opções globais da API Temporal Gisplay. Para cada opção tem a sua obrigatoriedade e o valor por omissão.

Vejamos agora na listagem 5.17 um exemplo completo onde o mapa de fundo é fornecido pela API Google Maps e o mapa dos Estados Unidos da América é adequadamente enquadrado. Neste exemplo não seria mostrado qualquer diálogo de progresso do carregamento dos dados.

```
let globalOptions = {
    showLoader: false,
    bounds: {
        usa: {
            NE: { lng: -67.631836, lat: 48.57479 },
            SW: { lng: -123.793945, lat: 23.563987 }
        }
   },
   provider: 'GM'
}
```

Listagem 5.17: Exemplo de fornecimento das opções globais

Eis, um exemplo completo do fornecimento das opções:

Opção	Obrigatoriedade	Default
bounds	Obrigatória	-
layout	Opcional	-
container	Obrigatória	-
provider	Opcional	Mapbox
showLoader	Opcional	true
mapOnClickFunction	Opcional	Popup
legendOnClickFunction	Opcional	-

Tabela 5.3: Resumo para cada opção global, qual a obrigatoriedade do seu fornecimento e o respetivo valor por defeito, se existir.

```
mapVariables: [{
        internalName: 'Género',
        externalName: 'gender',
        values: ["M", "F"]
    },
    {
        internalName: 'Idade do paciente',
        externalName: 'age',
        classBreaks: [0, 12, 19, 65, 81, 108]
    },
    {
        internalName: 'Latitude',
        externalName: 'latitudeParish',
    },
    {
        internalName: 'Longitude',
        externalName: 'LongitudeParish',
    }
],
optionalVariables: [{
        internalName: 'Dias de admissao',
        externalName: 'admissionDaysClass',
    },
    {
        internalName: 'Nome Hospital',
        externalName: 'desigHospital',
    }
],
urls: {dataURL: 'csvPortugalMultiple'},
time: {
    internalName: 'Ano dos casos de pneumonia',
    externalName: 'year',
    granularity: 'year',
}
```

Listagem 5.18: Exemplo completo de **parsingOptions**.

```
let mappingOptions = {
    color: {
        externalName: 'age'
    },
    shape: {
        externalName: 'gender',
        mapping: {
            "M": "filled_square",
            "F": "triangle",
        }
    }
};
```

Listagem 5.19: Exemplo completo de **mappingOptions**.

```
let globalOptions = {
    bounds: {
        pt: {
            NE: { lng: -4.954834, lat: 42.666281 },
            SW: { lng: -10.953369, lat: 35.406961 },
            description: 'Portugal Continental'
        }
    },
    container: 'gisplay',
    provider: 'MB',
    showLoader: true
};
```

Listagem 5.20: Exemplo completo de **globalOptions**.

Sendo a chamada feita da seguinte forma:

```
Gisplay.makeDotMap(parsingOptions, mappingOptions, globalOptions);
```

Listagem 5.21: Exemplo de chamada para criar o mapa de pontos respetivo

Na figura 5.1 é possível ver o mapa de pontos resultante das opções dadas nas listagens 5.18, 5.19 e 5.20 e da chamada ao *wrapper* de mapas de pontos na listagem 5.21. Neste mapa de pontos cada ponto representa um caso de pneumonia em Portugal Continental. Nesta figura é possível ver vários elementos que resultaram das opções fornecidas. Nomeadamente, a componente temporal que permite controlar vários instantes temporais, tendo cada um deles um ano diferente desde 2002 a 2011 e estando selecionado atualmente o ano de 2002. Nesta figura existem duas legendas, uma para cada variável visual. Sendo uma categórica que representa o género dos pacientes através da forma e outra contínua que representa a idade do respetivo paciente através da cor. Na figura 6.6b está representado o resultado do click num elemento do mapa. Neste caso, é apresentada a informação das variáveis opcionais e da variável contínua (idade), enquanto que o valor da variável categórica é facilmente perceptível pela forma pelo que não aparece no *picking*.

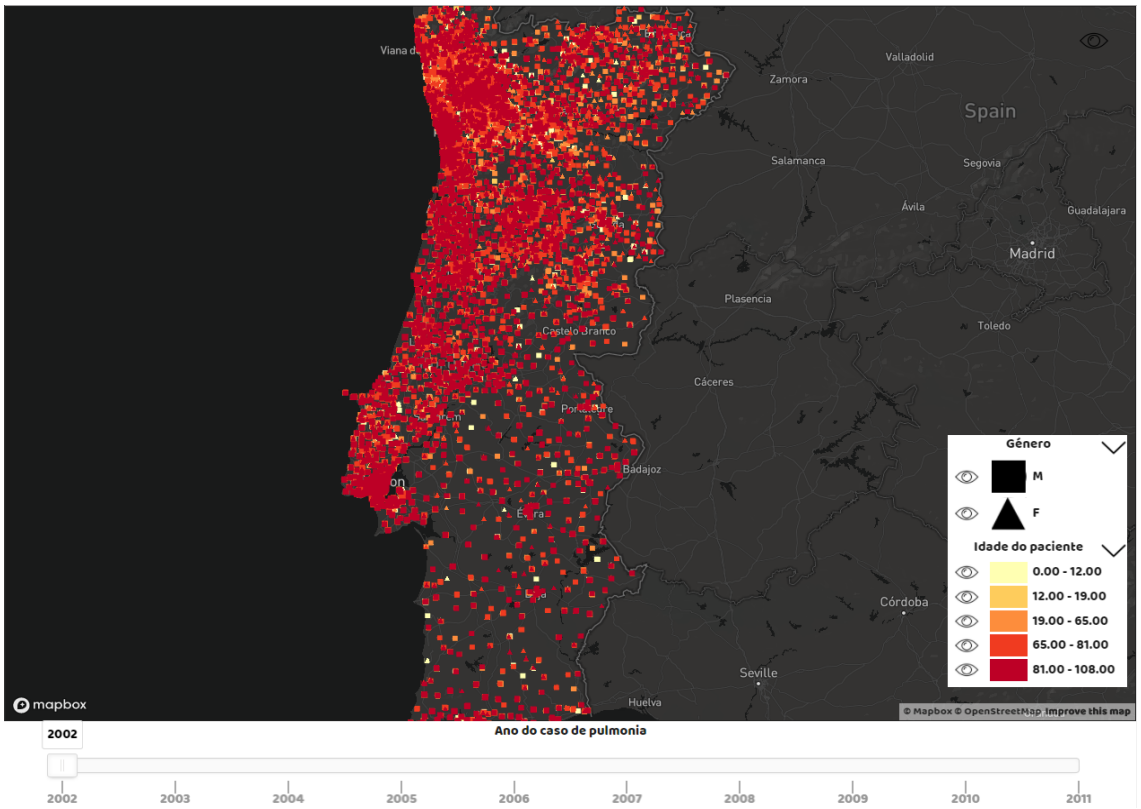


Figura 5.1: Mapa de pontos resultante das opções acima.

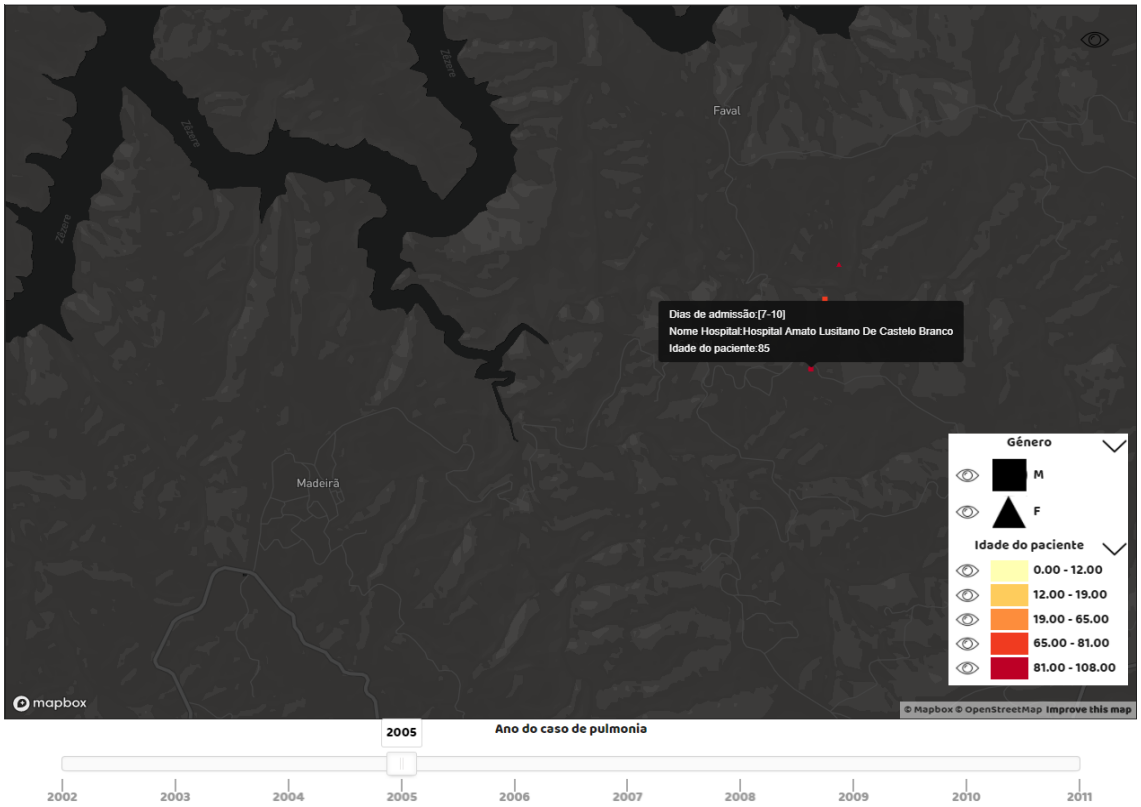


Figura 5.2: *Picking* no mapa de pontos.

5.2 Tratamento dos Dados

Todas as visualizações que venham a ser criadas com a API Gisplay, dependem como é óbvio, dos dados que sejam usados.

5.2.1 Parsers

Na API Gisplay existia um único *parser*, o *parser* para ficheiros *GeoJSON*. Este processava ficheiros no formato *GeoJSON*, e fazia-o lendo todo o conteúdo do ficheiro para memória e depois fazia o processamento através da função **JSON.parse()**[2]. Isto é possível pois o *GeoJSON* é um formato que segue as convenções do formato *JSON* permitindo que seja facilmente processado.

O único problema desta solução é que quando se tem um ficheiro com um tamanho considerável (e.g. 300 MB), ao tentar ler o ficheiro para memória de uma só vez vai na grande maioria das vezes levar a um *crash* da janela do browser. Na API Temporal Gisplay não foram feitas melhorias significativas neste *parser*, para além da melhoria do código que faz o processamento dos dados e da criação de uma classe específica que contém todo o código deste *parser*.

Com a adição da componente temporal verificou-se que a maioria dos *datasets* existentes tinham o formato CSV (*comma-separated values*), sendo como o nome indica um formato em que valores estão separados por um carácter, verificando-se na maioria das vezes que o carácter usado para a separação é a vírgula. No entanto, o carácter de separação entre colunas poderá ser ponto e vírgula (;) ou similar. O carácter de separação de linhas é um dos três: `\n`, `\r` ou `\r\n`, dependendo da forma e do sistema operativo no qual o *dataset* foi produzido.

O primeiro passo foi procurar que alternativas existem para processamento de ficheiros CSV *client-side* (no browser). Foram encontrados vários *parsers* que possibilitam o processamento dos ficheiros CSV:

- **jQuery-CSV**[3] - Faz processamento de CSV através de strings locais.
- **CSV.js**[4] - Tal como o anterior apenas faz processamento de CSV de string locais.
- **Papa Parse**[5] - Este é o *parser* de código aberto mais poderoso que existe atualmente para fazer processamento de ficheiros CSV *client-side*.

Dos três *parsers* acima mencionados, os dois primeiros apenas tratam de fazer processamento de strings, não fazendo diretamente de ficheiros locais. O terceiro *parser* é o mais poderoso dos três permitindo fazer processamento de ficheiros locais, de strings e

[2] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse
[3] https://github.com/evanplaice/jquery-csv
[4] https://github.com/knrz/CSV.js
[5] https://github.com/mholt/PapaParse

de ficheiros remotos, permitindo também que o ficheiro local seja lido em *stream*, isto é, lendo o ficheiro em várias partes para evitar que a aba do browser fique sem memória disponível. Como nesta dissertação precisamos de fazer processamento de ficheiros locais, então os dois primeiros foram logo à partida postos de parte pois para serem utilizados teríamos de fazer o processamento do ficheiro e fornecer a(s) string(s) resultante(s) ao *parser*.

O *Papa Parse* foi testado com vários datasets de grandes dimensões para testar se possuía a performance que pretendíamos. Os primeiros testes foram apenas para verificar o tempo de processamento do ficheiro, sem que seja guardado em memória qualquer informação dos valores lidos para cada coluna ou linha.

Tamanho do *dataset*	Número de linhas	Tempo de processamento
0.075GB	500K	1.5s
0.150GB	1M (1milhão)	2.8s
0.747GB	5M	12.3s
1.09GB	7.5M	18.4s
1.45GB	10M	24.7s
7.30GB	50M	168.0s

Tabela 5.4: Tempo de processamento de vários ficheiros com o *Papa Parse*.

Como podemos ver na tabela 5.4, para o processamento bruto de vários ficheiros é satisfatório para o que pretendemos nesta dissertação.

Depois foi testado, com um caso mais interessante e mais importante porque não foi testado o tempo de processamento bruto mas sim o tempo de processamento num caso real. Neste caso, usando os mesmos *datasets*, tendo estes 19 colunas e o número de linhas vistos na tabela 5.4. Das 19 colunas selecionaram-se duas colunas para a componente espacial (longitude, latitude), uma coluna para a componente temporal, duas colunas para atributos temáticos (duas variáveis, uma contínua e outra categórica) e uma coluna para uma variável opcional. Todas as colunas que forem guardadas é feito *cast* caso seja um número, permitindo diminuir a quantidade de memória necessária para guardar o valor lido e permite trabalhar depois com o número diretamente.

Tamanho do *dataset*	Número de linhas	Tempo de processamento
0.075GB	500K	2.0s
0.15GB	1M (1milhão)	4.3s
0.74GB	5M	42.8s
1.09GB	7.5M	Erro memória
1.45GB	10M	Erro Memória

Tabela 5.5: Tempo de processamento de vários ficheiros com o *Papa Parse*.

Como podemos ver na tabela 5.5, agora guardando as seis colunas, em que quatro delas são números e duas são strings, o *Papa Parse* teve uma diminuição significativa da performance. Para o caso com apenas 500 mil linhas a performance foi aproximadamente

75% da performance bruta (levou agora 2.0s contra os 1.5 anteriores). Para o caso com um milhão de linhas a performance foi ainda pior, sendo agora de apenas 65% da inicial, e no caso com cinco milhões de linhas a performance é de apenas 28% da inicial. A diminuição da performance, era expectável porque ao contrário do teste da performance bruta, neste teste estão a ser guardados algumas colunas dos dados. No caso com 7.5 milhões de linhas ou mais, acaba por dar erro sem memória porque está a guardar os dados do *dataset* e porque o *Papa Parse* lê a coluna completa e cria um objeto com cada chave do cabeçalho. O problema do erro por falta de memória revela-se quando se tem grande quantidade de dados que ultrapassa o limite imposto pelos browsers à quantidade de memória que uma aba possa usar, sendo de aproximadamente 2.5GB no *Chrome* e de aproximadamente 3GB no *Firefox*.

Visto isto, conseguimos perceber que o *Papa Parse* é satisfatório para *datasets* com um ou dois milhões de linhas. Para datasets de maiores dimensões devido à forma como está implementado, tem alguns problemas de performance e de memória utilizada. Foi então estudada a possibilidade de criar um novo *parser* CSV que use as mais recentes capacidades do browser (e.g. web workers[6]) e que resolva os problemas identificados para o *Papa Parse*.

O novo *parser* terá de ser capaz de:

1. Processar as linhas todas do ficheiro, ignorando as colunas de cada linha que não interessam para a visualização que pretendemos fazer.

2. Das colunas que fazem parte do problema, converter para o seu tipo (*cast*) e guardá-las como tal.

3. Processar o ficheiro de dados em paralelo, através do uso de *web workers*.

Com isto em mente foi criado um *parser* para a API Temporal Gisplay que funciona da seguinte forma:

1. A Thread principal lê e trata as opções do utilizador, nomeadamente quais as colunas do cabeçalho que interessam para a visualização.

2. A Thread principal divide o ficheiro em n partes e informa os vários *workers* qual a parte do ficheiro que devem ler e tratar de acordo com as opções dadas pelo utilizador.

3. Cada *web worker* lê a sua parte do ficheiro em bocados de 10MB e posteriormente faz o tratamento das linhas e das colunas, ignorando as que não interessam para o problema. Quando acaba de tratar cada pedaço de 10MB envia para a Thread principal o resultado do processamento.

4. A Thread principal ao receber cada pedaço de 10MB vai acondicionar os dados.

[6] https://developer.mozilla.org/pt-PT/docs/Web/API/Web_Workers_API

5. Quando os *web workers* chegam ao fim do processamento da sua parte enviam para a Thread principal a informação que falta processsar (primeira e última linha que leram), para evitar os casos em que a divisão entre *web workers* não aconteceu precisamente no carácter de mudança de linha.

6. A Thread principal envia as linhas que falta processar para um *web worker*, o qual faz o processamento das linhas remanescentes.

7. Quando a Thread principal receber as últimas linhas já processadas vai tratar de colocar toda a informação lida do ficheiro de acordo com a sua informação temática e do grânulo temporal a que pertencem.

O *parser* resultante desta implementação tem uma performance bem superior ao *Papa Parse*. Na seguinte tabela é possível ver a performance para o caso em que se dá o processamento bruto.

Tamanho do *dataset*	Número de linhas	Tempo de processamento (4 workers)
0.075GB	500K	0.95s
0.15GB	1M (1milhão)	1.6s
0.75GB	5M	8.6s
1.09GB	7.5M	12.2s
1.45GB	10M	16.5s
7.30GB	50M	82.7s

Tabela 5.6: Tempo de processamento bruto de vários ficheiros com o *Parser* da API Temporal Gisplay.

Em todos os casos vistos na tabela 5.6 o *parser* da API Temporal Gisplay tem uma performance superior ao *Papa Parse*, notando-se cada vez mais a diferença com o aumentar do número de linhas a processar.

Tamanho do *dataset*	Número de linhas	Tempo de processamento (4 workers)
0.075GB	500K	1.05s
0.15GB	1M (1milhão)	2.08s
0.75GB	5M	10.5s
1.09GB	7.5M	17.6s
1.45GB	10M	Erro memória

Tabela 5.7: Tempo de processamento real de vários ficheiros com o *Parser* da API Temporal Gisplay.

Como se pode ver na tabela 5.8, o *parser* da API Temporal Gisplay mantém uma performance linear, e consegue-o à custa do uso de múltiplo *web workers*. No exemplo acima foram usados quatro *web workers*.

Na tabela 5.8, é possível ver uma comparação mais completa entre o *Papa Parse* e o *parser* da API Temporal Gisplay. Na tabela temos várias linhas em que cada uma tem uma coluna com o tamanho do *dataset* em gigabytes, o número de linhas correspondentes

Tamanho	Linhas	**P.P.**	**G. (1w)**	(2w)	(3w)	(4w)	(5w)	(6w)	(7w)	(8w)
0.075GB	500K	2.0s	2.5s	1.4s	1.1s	1.1s	1.1s	1.2s	1.2s	1.2s
0.15GB	1M	4.3s	5.1s	3.0s	2.4s	2.1s	2.0s	2.0s	2.1s	2.2s
0.75GB	5M	42.8s	27.2s	14.7s	11.2s	10.3s	10.6s	10.7s	11.0s	11.2s
1.09GB	7.5M	Erro	37.6s	22.3s	17.2s	16.8s	17.0s	17.3s	17.8s	18.4s
1.45GB	10M	Erro	Erro	Erro	Erro	Erro	Erro	Erro	Erro	Erro

Tabela 5.8: Tempo de processamento real de vários ficheiros com o *Papa Parse* e com o *Parser* da API Gisplay. **P.P.=*Papa Parse***, **G.=Gisplay**, **1w=1 *web worker***. Os casos que contém o número seguido da letra w, representam o *parser* da API Gisplay com o respetivo número de workers.

e depois o tempo de processamento em segundos para o *Papa Parse* e para o *parser* da API Temporal Gisplay variando o número de *web workers* que processam o ficheiro em paralelo.

Com os resultados obtidos é possível perceber que usando apenas um *web worker* a performance da API Temporal Gisplay é 25% inferior em relação ao *Papa Parse* para o caso com 500 mil e um milhão de linhas, ao passo que para o caso com cinco milhões de linhas o *parser* da API Temporal Gisplay é 63% mais rápido. Esta mudança deve-se às características intrínsecas do *Papa Parse* que o levam a consumir maior quantidade de memória do que o *parser* da API Temporal Gisplay.

O *parser* da API Temporal Gisplay vai melhorando a performance em todos os *datasets* desde o caso em que usa um *web worker* até ao caso em que usa quatro *web workers*. Isto deve-se ao facto da máquina de testes possuir quatro cores, permitindo que os quatro façam a leitura e tratamento do ficheiro em paralelo. Nos testes efetuados a *Thread* principal apenas é responsável por receber os dados processados e guardar os mesmos em memória, pelo que não gasta quase tempo nenhum de processador. É de realçar o caso do *dataset* com cinco milhões de linhas, no caso em que o *parser* da API Gisplay usa quatro *workers* este é quatro vezes mais rápido do que o *Papa Parse*. O caso com dez milhões de linhas dá erro por falta de memória em ambos os *parsers* pois ultrapassa o limite que o browser suporta para cada aba.

O *parser* criado não implementa a especificação (RFC 4180[7]) na sua totalidade. Por exemplo, os casos em que se usem aspas não são tratados, isto porque para lidar com estes casos aumentaria a complexidade do *parser* e não traria ganhos significativos à API Temporal Gisplay. No entanto, se no futuro for necessário a implementação destes casos ou mesmo seguir por completo a especificação dos ficheiros CSV, acima mencionada, basta para tal modificar o código do método ***processRows(rows,...)*** da classe ***CSVDataWorker***. Como é óbvio vai diminuir a performance, nem que seja pela simples razão que tem de executar mais operações. No entanto, irá sempre beneficiar do uso da totalidade da capacidade de processamento do processador, o que não está a acontecer no *Papa Parse*.

[7] https://tools.ietf.org/html/rfc4180

5.3 Organização dos dados

Em termos de organização dos dados era necessário criar uma forma simples e que funcionasse para as várias primitivas geométricas, polígonos, pontos e linhas. A organização tem de ser simples porque os *browsers* impõem limites bastante restritivos na quantidade de memória que uma aba pode utilizar. Outro requisito da organização de dados é que não demore muito tempo a obter os dados para redesenhar, não comprometendo a interatividade e o *WebGL Picking* (ver secção 5.7).

Agora na API Temporal Gisplay, existe a possibilidade de ter várias variáveis visuais no mesmo mapa temático. Estas variáveis visuais podem ser todas categóricas, todas contínuas ou uma mistura de variáveis categóricas e contínuas. A API terá de ser capaz de lidar com um número indefinido de variáveis visuais ao mesmo tempo. Existe ainda a componente temporal, e num mapa temático podem existir várias variáveis visuais e vários grânulos temporais (instantes no tempo) em simultâneo. Tudo isto combinado leva a um acréscimo acentuado da importância da organização dos dados em estruturas compatíveis com os requisitos anteriormente apresentados.

Imaginando, um exemplo que existem duas variáveis visuais:

- a primeira é categórica com 2 categorias distintas para o género (“Male”, “Female”)
- a segunda contínua com três classes distintas para a idade ([0, 25[, [25, 65[, [65, 108])

Uma maneira de identificar as classes e as categorias é através de um índice começado em zero para cada uma das variáveis visuais.

No exemplo anterior, para a variável categórica teríamos 0→“Male” e 1→“Female”.

Para a contínua teríamos [0, 25[→0, [25, 65[→1 e [65, 108]→2.

Então, podemos fazer o produto cartesiano dos índices das duas variáveis visuais, resultando em:

- “Male”→0 e [0, 25[→0, resulta em (0, 0)
- “Male”→0 e [25, 65[→1, resulta em (0, 1)
- “Male”→0 e [65, 108]→2, resulta em (0, 2)
- “Female”→1 e [0, 25[→0, resulta em (1, 0)
- “Female”→1 e [25, 65[→1, resulta em (1, 1)
- “Female”→1 e [65, 108]→2, resulta em (1, 2)

Para o resultado de cada elemento do produto cartesiano foi criada a classe *MapVariableCombination*. Sendo assim, todos os dados que digam respeito a cada combinação

resultante irão para a respetiva *MapVariableCombination*. Por sua vez a classe *MapVariableCombination* terá um ou mais grânulos temporais, sendo cada um representado pela classe *TemporalGranule*.

Antes de entrar em detalhe na classe *MapVariableCombination* é preciso entender como é que foi estruturada a forma de guardar informação relativa às variáveis que são dadas inicialmente pelo programador. Como foi visto na secção 5.1 (opções da API Gisplay), o programador pode fornecer opções para: variáveis mapeadas em variáveis visuais, variáveis opcionais, variáveis geográficas (latitude e longitude) e variável relativa à componente temporal.

Todas as variáveis anteriormente referidas têm em comum o nome interno e externo, pelo que se criou a classe base chamada *DataVariable* que tem os referidos valores e pode ser facilmente estendida. Esta classe pode ser usada para as identificar as variáveis geográficas pois estas só possuem nome interno e externo. O nome interno é reservado na API para cada uma delas, sendo "Latitude" e "Longitude", respetivamente. Por outro lado, pode ser também usada para as variáveis opcionais, pois nestas só é possível fornecer o nome interno e externo.

Depois temos a *TimeVariable*, que estende a classe *DataVariable* e permite que se guarde informação da variável relativa à componente temporal. Nesta informação destacam-se o nome interno e externo já presentes na classe base, a granularidade e quais os valores associados a esta granularidade, sendo que estes valores podem ser fornecidos inicialmente pelo programador ou calculados no momento da leitura dos dados.

Para as variáveis mapeadas em variáveis visuais no mapa temático, visto que têm várias coisas em comum além do nome interno e externo, foi então criada a classe *MapVariavle*, que como era de esperar estende a classe base *DataVariable*. O que existe em comum nas variáveis mapeadas em variáveis visuais são os valores, o nome de variável visual e qual o mapeamento de cada categoria ou classe para um valor da variável visual (e.g. "Male" mapeado pela cor azul).

Cada um dos tipos de variáveis terá a sua classe, sendo que para as variáveis categóricas existe a classe *CategoricalVariable* e para as contínuas a classe *ContinuousVariable*, e que como já foi dito ambas estendem a classe *MapVariable*.

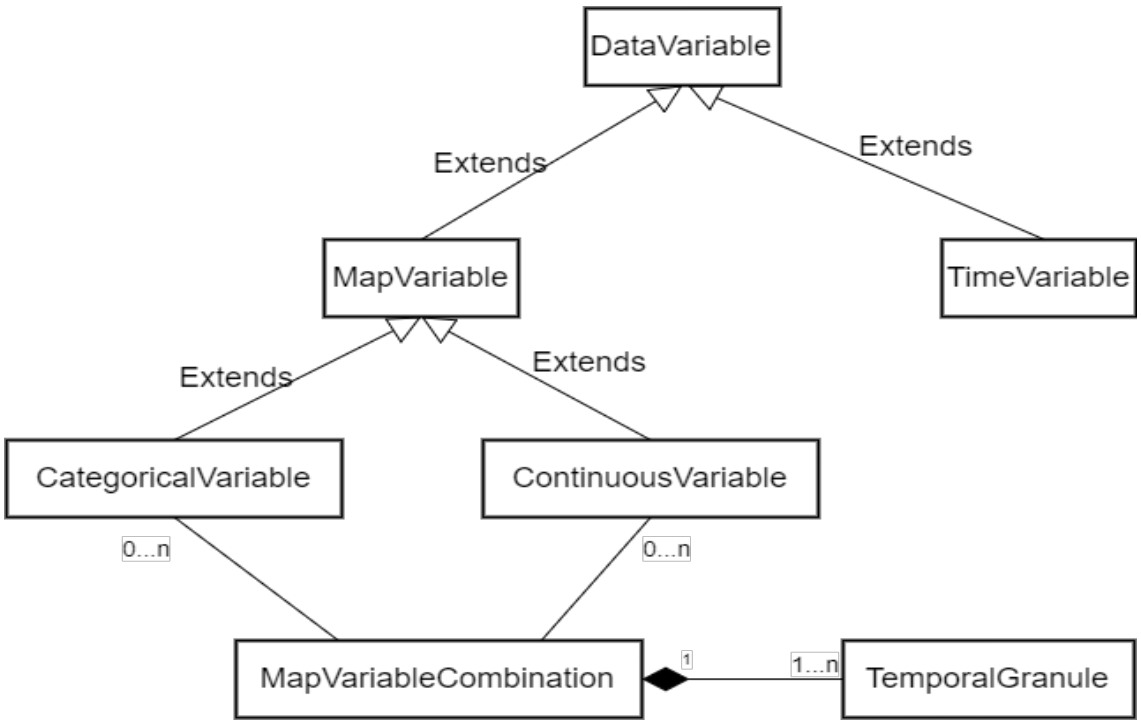


Figura 5.3: Diagrama de classes para as variáveis temporal, opcionais.

Na figura 5.3 é possível ver um diagrama que descreve as relações entre as classes acima descritas. Como já foi dito cada combinação, representada por uma classe *MapVariableCombination* pode ter um ou mais grânulos temporais (*TemporalGranule*). Por outro lado, cada *MapVariableCombination* é constituída por variáveis contínuas (*ContinuousVariable*) ou por variáveis categóricas (*CategoricalVariable*) ou por ambas.

Os *parsers* que existem ou que venham a existir no futuro têm de fornecer os dados para a *MapVariableCombination* de acordo com uma estrutura bem definida. Esta estrutura tinha de ser o mais simples possível, para que no caso em que se tem de calcular as classes de uma ou mais variáveis contínuas ou que os diferentes grânulos temporais não tenham sido fornecidos pelo programador, permita que sejam guardadas temporariamente milhões de linhas de dados.

Durante o *parsing* dos dados, é muitas vezes necessário guardar os dados processados temporariamente e só no fim deste processamento é que se pode fazer a sua distribuição pelas respetivas *MapVariableCombination*. Isto acontece quando temos pelo menos uma variável contínua com os limites das classes por calcular, pois desta forma ainda não é possível fazer a separação de cada elemento lido pois o mesmo ainda não tem uma classe atribuida.

Para guardar os dados temporariamente foi pensada uma estrutura que guarda cada linha dos dados num *array*, da seguinte forma:

$$temporaryLine = [[\underbrace{[VariaveisMapeadas]}_{0}, \underbrace{[Geometria]}_{1}, \underbrace{[Tempo]}_{2}, \underbrace{[VariaveisOpcionais]}_{3}]]$$

Para cada posição de uma *temporaryLine*:

0. Informação relativa às variáveis que serão mapeadas em variáveis visuais no mapa temático. As variáveis categóricas já estarão guardadas de acordo com o seu índice enquanto que para as contínuas estará guardado o valor que foi lido diretamente dos dados. Fazendo logo a conversão das variáveis categóricas para o respetivo índice consegue-se poupar memória pois basta guardar um número em vez de uma *string*. A ordem vai ser primeiro as variáveis categóricas, depois as variáveis contínuas que não precisam de cálculo de classes e por fim as variáveis contínuas que necessitam de cálculos de classes.

1. A informação relativa à geometria, sendo que esta pode ser um ponto, um polígono, uma linha ou um identificador caso a geometria seja reutilizada.

2. Índice do grânulo temporal. Tal como existe para as combinações, cada grânulo temporal tem um índice que o identifica. Os identificadores são gerados sequencialmente a partir de zero pela ordem de leitura.

3. Valores das variáveis opcionais. Caso exista mais do que uma variável opcional a sua ordem vai ser a ordem como foram tratadas inicialmente nas opções.

Depois de calcular as classes para as variáveis contínuas que necessitavam das suas classes calculadas basta usar o índice da combinação para decidir em que *MapVariableCombination* se deve adicionar e com o índice temporal é possível na combinação determinar qual o grânulo temporal a que a linha de dados pertence. Esta forma de guardar os dados permite que se use o mínimo de memória possível e que a distribuição dos dados seja o mais rápida possível.

A estrutura e organização dos dados anteriormente apresentada é também escalável independentemente do número de variáveis mapeadas em variáveis visuais.

5.4 Cálculo de classes

Com os datasets do mundo real a quantidade de dados é cada vez maior. Para fazer o cálculo de classes para milhões de elementos muitas das vezes o tempo consumido para fazer este cálculo é proibitivo.

Na versão anterior da API Gisplay, era usada a API chromajs para a escolha das cores e também para o cálculo dos limites das classes das variáveis contínuas. Estes cálculos usando todos os valores que existem para determinada variável são os mais corretos mas

sofrem de um grande problema de performance para conjuntos de dados que tenham vários milhões de pontos (e.g. 10 milhões de pontos).

Testando um exemplo simples para o cálculo de classes com a API chromajs com dez milhões de inteiros gerados de forma aleatória, é possível verificar que a performance não é a desejada. Para o método de intervalos iguais (*equal intervals*) demorou aproximadamente 3.5 segundos, para o método de quantiles aproximadamente 3.6 segundos e para o método k-means quase 1 minuto (58.9 segundos). Como é possível verificar por este teste, é ideal encontrar uma forma que torne este cálculo para qualquer um dos métodos acima, ou outros, bem mais rápido do que os valores apresentados.

Visto isto, para o cálculo das classes das variáveis contínuas foi proposta uma solução baseada num histograma. Para tal, vamos precisar dos valores mínimo e máximo (dados pelo programador nas opções da variável contínua) e de um número de *bins*. Os *bins* são os intervalos pelos quais o histograma estará dividido. Cada *bin* terá um intervalo de valores, sendo que todos os intervalos têm o mesmo tamanho.

Sabendo que, o objetivo é construir classes para mapas temáticos e quem fará apreciação e interpretação destes mapas temáticos são humanos, os quais têm limitações cognitivas na perceção de elementos gráficos, então o número de *bins* terá de ter isto em conta. Sendo assim, oito classes diferentes será normalmente já um caso extremo para a perceção humana e mais do que isto causará ainda mais confusão.

Com as premissa do parágrafo anterior foi proposto que o histograma tenha cem *bins*. Com cem *bins* tem assim um valor bem maior que as oito classes do caso extremo anteriormente referido, sendo então possível criar as classes.

O cálculo das classes passará então a fazer-se da seguinte forma:

1. Cálculo dos intervalos de cada *bin* através do valor mínimo e máximo e do número de *bins* (cem).

2. Para cada linha dos dados recebidos, verificar o valor da coluna para a qual se pretende fazer o cálculo das classes e consequentemente somar um ao intervalo no histograma a que esse valor pertença.

3. No fim da leitura e tratamento dos dados, e tendo o histograma, calcula-se então as classes usando um dos algoritmos fornecidos pela API Gisplay.

Os algoritmos fornecidos para o cálculo de classes (*quantiles*, *equal intervals*, etc) passarão a ser algoritmos aproximados, mas como temos vários milhões de elementos esta aproximação é razoável e neste contexto aceitável. Mais concretamente, os algoritmos serão aproximados porque as linhas de corte vão passar a ser os *bins* em vez dos valores concretos.

Com esta alteração, o tempo para o cálculo de classes é reduzido significativamente e acima de tudo é escalável com o aumento do número de valores, pois só tem de guardar cem valores inteiros (um para cada intervalo do histograma), independentemente do número de linhas que sejam lidas dos dados.

5.5 Mapas de Fundo

Na API Gisplay apenas se dava acesso a um fornecedor de mapas de fundos, o *Mapbox*[8]. Segundo o documento final da API Gisplay, para adicionar novos mapas de fundo seria necessário estender a classe *BGMapWrapper*. Veio-se a verificar não ser exatamente assim ao testar a API *Google Maps*[9], pois existiam várias diferenças entre as *APIs* em termos de nomes de eventos e o seu retorno

Sendo assim, anteriormente a classe *BGMapWrapper* era usada para o fornecedor *Mapbox*, passando agora a ser usada para conter os métodos semelhantes a todos os mapas de fundo que venham a entender esta classe. Para o *Mapbox* criou-se a classe *BGMapMapBox* e para o *Google Maps* a classe *BGMapGoogleMaps*.

O novo *wrapper*(*BGMapWrapper*) possui os métodos que não necessitam de ser implementados pelas classes que o estendam e possui outros métodos abstratos que terão, como é óbvio, de ser implementados por todo e qualquer mapa de fundo que implemente esta classe.

A classe *BGMapWrapper* tem os seguintes métodos implementados:

- **getWidth()** - Retorna a largura em pixeis do elemento na DOM que possui o mapa de fundo.
- **getHeight()** - Retorna a altura em pixeis do elemento na DOM que possui o mapa de fundo.
- **calculateViewPort(canvas)** - Calcula o *viewport* do mapa de fundo dentro do elemento *canvas*.
- **calcNextPowerOfTwo()** - Calcula para a altura e para a largura qual é o valor seguinte que é elevado a 2, sendo usado para as texturas.
- **getTileSize()** - Retorna qual o tamanho em pixeis de cada imagem do mapa de fundo. Todos os mapas de fundo que foram adicionados usam o valor 256 por defeito. Se existir algum mapa que venha a ser adicionado em que este valor difira então terá de ser a classe que estender esta a declarar o novo valor.

Por outro lado, a classe *BGMapWrapper* tem os seguintes métodos abstratos:

- **createBackgroundMap(mapId, bounds)** - Instancia o mapa de fundo, associando esta instância ao elemento na DOM que tem o identificador *mapId* e com os limites da vista iguais aos que forem dados nos *bounds*.
- **getZoom()** - Retorna o nível de zoom do mapa de fundo.
- **getCenterLng()** - Retorna a longitude do centro.

[8] https://www.mapbox.com/mapbox.js/api/v3.1.1/
[9] https://developers.google.com/maps/documentation/javascript/

- **getCenterLat()** - Retorna a latitude do centro.
- **addEventListener(eventstr, eventfunction)** - Associa ao evento do mapa de fundo com nome *eventstr* a função *eventfunction*, sendo que esta função será chamada cada vez que o evento ocorra.
- **addPanEvent(panFunction)** - Adiciona o evento de *pan* ao mapa de fundo, sendo que cada vez que este ocorrer a função *panFunction* será chamada.
- **addZoomEvent(zoomFun)** - Adiciona o evento de *zoom* ao mapa de fundo, sendo que cada vez que este ocorrer a função *zoomFun* será chamada.
- **addClickEvent(clickFunction)** - Adiciona o evento de *click* ao mapa de fundo, sendo que cada vez que este ocorrer a função *clickFunction* será chamada.

Estes métodos abstratos estão assim implementados nos *wrappers* que estendam a classe *BGMapWrapper*. Foram implementados na API Temporal Gisplay os seguintes mapas de fundo: *Google Maps*, *Here Maps*[10], e *Bing Maps*[11], aos quais se junta o já existente *Mapbox*.

5.5.1 Matriz de projeção

Ao ser feita esta alteração e adição de novos mapas de fundo foi detetado um erro no desenho nos novos mapas usados que provém da forma como os mesmos reportam o valor da longitude do centro. O mapa de fundo *Mapbox* reportava valores entre -180 e +180 para a longitude, enquanto que o *Google Maps* pode reportar valores menores do que -180 ou maiores do que +180 se fizermos *pan* na vista. Outro "problema" identificado, era a conversão de coordenadas geográficas para coordenadas *canvas* a ser feita inicialmente pelo processador. Este caso era um problema porque não poupava qualquer espaço pois o número guardado desta conversão continuava a ocupar 8 bytes (como qualquer número em javascript), ainda gastava tempo significativo com esta conversão ao executar em milhões de ciclos do processador e acima de tudo porque esta conversão pode ser feita em paralelo pela placa gráfica (WebGL) usando a matriz de projeção correta.

Como a conversão inicial para coordenadas *canvas* não é necessária, então esta deixa de ser feita, sendo ao invés guardadas as coordenadas geográficas tal como estas são lidas do *url* de dados. Depois, a matriz de projeção terá de possibilitar a conversão de coordenadas geográficas para coordenadas *canvas* e destas para coordenadas WebGL, sendo que na API Temporal Gisplay todos os mapas fazem estes cálculos da coordenada final através desta matriz de projeção no *vertex shader* do mapa.

Primeiro, para converter de coordenadas geográficas para coordenadas *canvas*, é preciso converter de (longitude, latitude) para pixeis (x, y). As coordenadas *canvas* começam em (0,0) no canto superior esquerdo e vão até (largura, altura) no canto inferior direito.

[10] https://developer.here.com/documentation/maps/topics/quick-start.html
[11] https://msdn.microsoft.com/pt-pt/library/dd877180.aspx

Como vimos na subsecção 2.1.1, a projeção é a conversão das três dimensões da superfície real do globo para as duas dimensões do mapa (superfície plana). Todos os mapas de fundo que foram adicionados à API Temporal Gisplay usam a projeção *Web Mercator*, sendo também esta a usada por praticamente todos os outros mapas de fundo existentes[11].

A fórmula para a projeção *Web Mercator* é:

$$x = (\frac{128}{\Pi})2^{zoom}(\lambda + \Pi)pixels$$

$$y = (\frac{128}{\Pi})2^{zoom}(\Pi - ln[\tan(\frac{\Pi}{4} + \frac{\phi}{2})])pixels$$

Onde ϕ é a longitude de um ponto em radianos e λ é a latitude de um ponto em radianos.

Depois de convertido para pixeis do *canvas* é preciso converter para coordenadas WebGL, sendo que as mesmas variam entre (-1, -1) no canto inferior esquerdo e (1, 1) no canto superior direito, sendo (0,0) no centro. Esta conversão é feita através da normalização do valor (x, y) obtido para o intervalo [0,2] e depois para [-1,1].

Cada vez que se redesenha sobre o mapa de fundo, tem primeiro de se afetar a matriz de projeção que será depois usada para calcular a posição do ponto/vértice através das fórmulas vistas acima.

As matrizes resultantes da fórmula acima são:

$$\underbrace{\begin{bmatrix} \frac{2}{width} & 0 & 0 & -1 \\ 0 & \frac{2}{height} & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{M1} \underbrace{\begin{bmatrix} scale & 0 & 0 & offsetX \\ 0 & scale & 0 & offsetY \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{M2} \underbrace{\begin{bmatrix} \frac{\Pi}{180} & 0 & 0 & \Pi \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{M3} \underbrace{\begin{bmatrix} Longitude \\ \Pi - ln[\tan(\frac{\Pi}{4} + \frac{\phi}{2})] \\ 0 \\ 1 \end{bmatrix}}_{M4}$$

Onde *width* e *height* são a largura e altura, respetivamente, do elemento DOM que contém o mapa de fundo, *scale* é a escala calculada a partir do tamanho de cada imagem do mapa de fundo e do nível de zoom e *offsetX*/*offsetY* são a diferença em x e y desde o centro. Todas estas variáveis são afetadas pelo processador.

A matriz M2, M3 e M4 servem para fazer conversão de coordenadas geográficas para coordenadas *canvas*, sendo que a matriz M1 serve depois para normalizar de coordenadas *canvas* para o intervalo [0,2] e depois deste para o intervalo [-1,1] (do WebGL).

A multiplicação da matriz M1,M2 e M3 é uma multiplicação, pelo que pode ser simplificada numa só matriz. Tendo H = *height*, W = *width*, S = *scale*, X = *offsetX* e Y = *offsetY*, temos:

$$\underbrace{\begin{bmatrix} \frac{\Pi \cdot S}{90 \cdot W} & 0 & 0 & \frac{2 \cdot S \cdot \Pi + 2 \cdot X - W}{W} \\ 0 & \frac{2*S}{H} & 0 & \frac{2*Y}{H} - 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{Mx} \underbrace{\begin{bmatrix} Longitude \\ \Pi - ln[\tan(\frac{\Pi}{4} + \frac{\phi}{2})] \\ 0 \\ 1 \end{bmatrix}}_{M4}$$

Assim, o processador faz a afetação da matriz Mx para cada redesenho e a placa gráfica trata de paralelamente fazer fazer a multiplicação de Mx por M4 para cada ponto/vértice e assim obter as coordenadas WebGL respetivas.

Com esta solução, deixou de existir a conversão de coordenadas geográficas para coordenadas *canvas* logo não se gasta tempo desnecessário de processador e é também uma solução que funciona para todos os mapas de fundo adicionados à API Temporal Gisplay bem como qualquer outro que use a projeção *Web Mercator*.

5.6 Múltiplas Projeções

As múltiplas projeções são muito importantes para visualizar vários locais do globo que estão fisicamente distantes ou para visualizar uma determinada região mais detalhadamente.

Primeiro, é necessário criar uma forma de possibilitar ao programador o fornecimento das opções para as múltiplas projeções e qual a forma como estas estão distribuídas no ecrã (*layout*).

5.6.1 Definição das regiões a visualizar

O fornecimento das múltiplas projeções é a parte mais fácil deste problema, bastando para tal dar os limites das projeções pretendidas. Estes limites são dois cantos opostos. Os pontos poderão ser o nordeste e sudoeste ou o noroeste e sudeste, pois a partir dos dois primeiros é possível chegar aos dois últimos e vice-versa.

O fornecimento dos limites será feito da seguinte forma: para cada projeção pretendida, fornecer as coordenadas do ponto nordeste (**NE**) e sudoeste (**SW**) e a descrição (**description**) dessa projeção.

No exemplo 5.22 é possível ver o fornecimento das projeções para o território continental dos Estados Unidos, Alaska e ilhas do Havaí. Para cada um dos pontos (nordeste e sudoeste) de cada projeção é fornecida a respetiva longitude e latitude. A descrição será usada para identificar a vista na visualização final. A latitude e longitude de cada limite é fornecida na forma de chave-valor, mas poderia ser fornecido na forma de *array* ou mesmo fornecer os dois pontos limite num único *array*, em que os dois primeiros elementos eram respetivos à latitude e longitude do ponto sudoeste e os dois últimos elementos, respetivos à latitude e longitude do ponto nordeste. Em qualquer um dos casos, só existe a necessidade de seguir sempre o padrão escolhido para que a API funcione

corretamente. A forma escolhida (chave-valor), é mais verbosa que na forma de *array*, no entanto é bastante mais fácil perceber qual a correspondência de cada valor fornecido.

```
bounds: {
    usa: {
        NE: { lng: -67.631836, lat: 48.57479 },
        SW: { lng: -123.793945, lat: 23.563987 },
        description: 'Mainland USA'
    },
    alaska: {
        NE: { lng: -138.427734, lat: 72.289067 },
        SW: { lng: -174.638672, lat: 50.176898 },
        description: 'Alaska'
    },
    hawaii: {
        NE: { lng: -162.388916, lat: 17.056785 },
        SW: { lng: -153.4021, lat: 22.917923 },
        description: 'Hawaii'
    }
}
```

Listagem 5.22: Exemplo de fornecimento dos limites das várias projeções

5.6.2 Definição do *layout*

Tendo sido definida a forma de especificar as várias regiões do mapa, era necessário encontrar uma forma fácil e intuitiva para fornecer o *layout* através do qual as múltiplas projeções estarão dispostas.

O processo para criação do *layout* passa por aplicar um algoritmo recursivo de divisão. Em cada nível escolhe-se uma direção (horizontal ou vertical) e divide-se o espaço em fatias do tamanho relativo dado. Cada uma das fatias do espaço é passível de ser subdividida usando o mesmo processo.

Sendo assim, foi proposta a seguinte especificação para o layout:

1. Fornecer qual o corte principal (vertical ou horizontal) para o elemento DOM que contém a instância corrente da API Gisplay.

2. Para o corte principal ou qualquer outro corte, irão existir dois filhos:

    a) *sizes* - qual o tamanho de cada um dos elementos do corte.

    b) *descendants* - quais são os descendentes do elemento atual. Existirão sempre tantos elementos-filho (descendentes) quantos os elementos do *array sizes*. Os descendentes podem ser: uma folha, representado por uma string com o identificador de um dos limites das projeções (fornecidos nos *bounds*), ou outro elemento do tipo vertical/horizontal. No caso do elemento ser uma folha então é criado na DOM um elemento com o tamanho especificado, caso seja outro elemento vertical/horizontal é repetido o ponto 2.

```
layout: {
    vertical: {
        sizes: [40, 60],
        descendants: [
            {
                horizontal: {
                    sizes: [60, 40],
                    descendants: ["alaska", "hawaii"]
                }
            },
            "usa"
        ]
    }
}
```

Listagem 5.23: Exemplo de fornecimento do layout para os limites do exemplo 5.22

No exemplo 5.23 é possível ver um exemplo do fornecimento do *layout* para os limites das projeções anteriormente apresentadas. Os limites do exemplo 5.22 em conjunto com o *layout* do exemplo 5.23 dão origem às projeções mostradas na imagem 5.4.

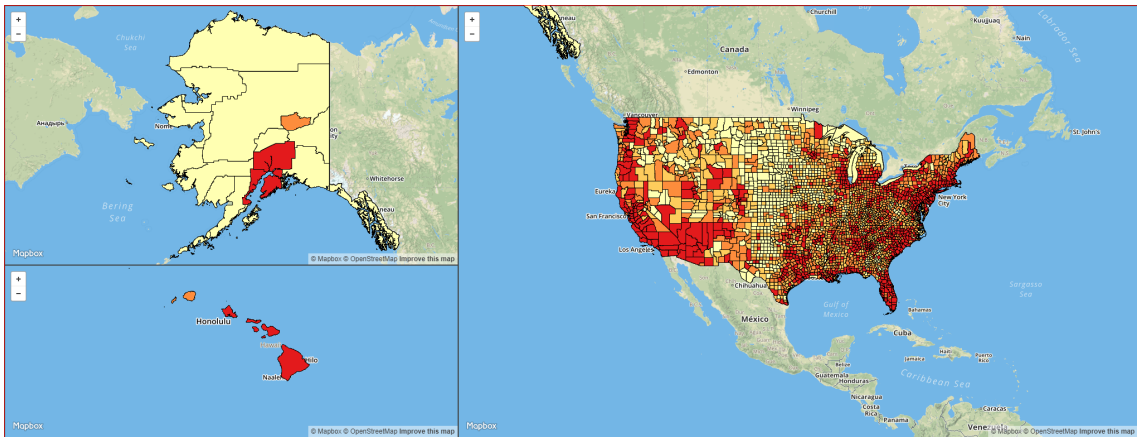


Figura 5.4: Múltiplas projeções criadas pelas *bounds* e *layout* vistos anteriormente.

A adição das múltiplas projeções levou a algumas modificações fundamentais na API Gisplay. Anteriormente, só era possível ter uma vista sobre o mapa temático, logo existia apenas um elemento *canvas* com a referida vista. Com várias projeções existem duas possibilidades, a primeira é ter um *canvas* para cada vista ou ter um único *canvas* dividido em *viewports*.

A primeira hipótese levava a que os dados (pontos/polígonos) necessitassem de ser divididos para a(s) vista(s), nas quais estavam inseridos. Isto resultaria em múltiplos contextos *WebGL*, um contexto para cada *canvas* existente. Resultaria também no aumento do espaço necessário para guardar os dados, visto que existiria repetição dos mesmos.

A segunda hipótese passa por ter um único *canvas*, logo um único contexto *WebGL* e dividir o *canvas* em vários *viewports*, um *viewport* para cada vista.

A solução escolhida foi a segunda porque esta mantém a simplicidade que já existia na versão anterior da API Gisplay e evita criar múltiplos contextos *WebGL*. Para fazer o desenho em cada uma das projeções é feito o desenho de todos os dados que estejam ativos no momento, ficando ao cargo do *WebGL* o desenho apenas daqueles que estão dentro dos limites do respetivo *viewport*.

No futuro, se houver necessidade, poderá ser usada uma estrutura de dados que permita fazer a divisão dos dados por cada uma das projeções, permitindo desta forma que quando exista um evento que só se reflita numa das projeções só é necessário redesenhar os dados que dizem respeito à mesma e que os dados estejam devidamente separados pela(s) vista(s) a que dizem respeito.

5.7 Picking

Na API Gisplay, o evento *click* num elemento do mapa era suportado por duas estruturas de dados diferentes. Para os polígonos era usada a *RBush*[12], que é uma versão otimizada da conhecida *rtree*[28], enquanto que para os pontos era usada a *kdtree*[13]. Estas estruturas tem a forma de árvore e permitem que a procura de um elemento no espaço seja extremamente rápida. Os principais problemas destas estruturas são o tempo inicial para a sua criação e o seu custo em termos de memória ocupada. Em termos de tempo inicial de criação, a *rtree* gastava mais de 30% do tempo inicial da API Gisplay nos mapas de polígonos, enquanto que a *kdtree* gastava muitas das vezes mais do que 80% do tempo inicial para os mapas de pontos. Em termos de memória ocupada, a *rtree* usava todo o conteúdo do *GeoJSON* lido. A *kdtree* permite que o programador forneça a função de distância, sendo esta usada para determinar quais os pontos que estão mais próximos do local do *click* no mapa.

Visto isto, como os problemas das estruturas estão identificados, era necessário procurar uma ou mais soluções para estes problemas. Em vez de se procurar outras estruturas do mesmo tipo foi tido em conta a possibilidade de usar o *WebGL* e as suas capacidades para dar suporte ao *click* num elemento do mapa.

O *WebGL* não tem nativamente suporte para o *click* num elemento do mapa, mas podem ser usadas as suas capacidades para o mesmo efeito. Nomeadamente, é possível implementar o *WebGL Picking* também conhecido como *hit testing*, através da codificação de cores[14]. Para tal, cada elemento existente terá uma cor *RGBA* única que servirá depois para a identificação do mesmo quando se dá o *click* no mapa.

Um dos grandes objetivos desta dissertação é que a API Temporal Gisplay seja capaz de lidar com grandes volumes de dados, na ordem dos vários milhões de pontos. Para tal, se existem milhões de pontos, precisamos também de identificar cada um deles usando uma cor diferente. Com codificação por cores *RGBA* temos exatamente **4 294 967 296**

[12] https://github.com/mourner/rbush
[13] https://github.com/ubilabs/kd-tree-javascript
[14] https://github.com/FarhadG/webgl-picking/blob/master/colorEncoding.js

hipóteses diferentes, querendo isto dizer que poderíamos ter este número de elementos diferentes no mapa. Hoje em dia, o browser em termos de memória, não tem capacidade para lidar com muitos milhões de pontos logo esta solução funciona perfeitamente para as capacidades correntes e para as possíveis capacidades futuras da API Temporal Gisplay.

Como foi visto na secção 5.3, são criadas as combinações de variáveis mapeadas por variáveis visuais. Num caso extremo com quatro variáveis mapeadas por variáveis visuais e em que cada uma destas possui oito classes/categorias diferentes, teríamos 8^4 combinações possíveis. Sabendo que, o número de cores diferentes *RGBA* é 256^4 e o número de combinações do exemplo anterior são 8^4, então temos **1048576** cores para cada uma das combinações existentes, o que é perfeitamente aceitável.

A geração de identificadores para cada elemento funciona da seguinte forma:

- Divisão do intervalo de cores possíveis pelo número de combinações existentes. Cada combinação de variáveis mapeadas por variáveis visuais terá um número de identificadores igual a todas as outras combinações, ou seja, divide o intervalo de identificadores possíveis em intervalos iguais.

- Quando se cria uma combinação atribui-se o respetivo identificador *RGBA* inicial. A primeira combinação terá sempre identificador inicial igual a (0, 0, 0, 1), porque a combinação *RGBA* (0, 0, 0, 0) é reservada para representar o *click* que não tenha sido num elemento dos dados.

- Para cada elemento que se vá adicionar num determinado grânulo temporal de uma combinação, gera-se um identificador sequencial a partir do respetivo identificador inicial.

Num exemplo em que só exista uma variável categórica com três categorias diferentes, logo existirão três combinações possíveis (uma para cada uma das categorias), o identificador inicial da primeira combinação será (0, 0, 0, 1), da segunda (85, 85, 85, 86) e o da terceira será (170, 170, 170, 171). Para a primeira combinação os valores possíveis variam sequencialmente entre (0, 0, 0, 1) e (85, 85, 85, 85), para a segunda de (85, 85, 85, 86) até (170, 170, 170, 170), enquanto que na terceira vão desde (170, 170, 170, 171) até (255, 255, 255, 255). Para este exemplo podemos ver na tabela 5.9 o valor inicial de cada um das combinações.

Identificador *RGBA* inicial	**Valor inicial**
(0, 0, 0, 1)	$256^3 \times 0 + 256^2 \times 0 + 256^1 \times 0 + 1 = 1$
(85, 85, 85, 86)	$256^3 \times 85 + 256^2 \times 85 + 256^1 \times 85 + 86 = 1411655766$
(170, 170, 170, 171)	$256^3 \times 170 + 256^2 \times 170 + 256^1 \times 170 + 171 = 2863311531$

Tabela 5.9: Identificador *RGBA* de cada combinação e respetivo valor inteiro.

Depois dos dados estarem corretamente acondicionados e todos terem os seus respetivos identificadores, temos de ser capazes de ao clicar num elemento do mapa temático

procurar a sua informação. Para tal, quando se dá o *click*, o mapa de fundo clicado irá lançar o evento respetivo, sendo que este evento possui as coordenadas (x,y) do pixel clicado. Depois disto, faz-se o verdadeiro *WebGL Picking*, isto é, desenha-se o mesmo conteúdo que está no mapa de fundo onde se deu o evento, mas em vez de se desenhar no ecrã desenha-se numa textura. Este desenho é praticamente igual ao que se faz para desenhar no mapa temático, a principal diferença é que a cor utilizada é o identificador *RGBA* respetivo a cada elemento que for desenhado.

O processo de desenho na textura é o seguinte:

1. Obter a localização do pixel (x,y) clicado no mapa de fundo.
2. Desenhar os dados para uma textura em vez do ecrã.
3. Obter da textura o valor *RGBA* do pixel clicado.

Depois de obtido o valor do pixel clicado na textura, tem de se procurar qual o respetivo elemento. Esta procura tem de ser o mais rápida possível, de modo a ter performance semelhante às estruturas (*rtree* e *kdtree*) utilizadas na versão anterior da API Gisplay.

O primeiro passo é determinar em qual combinação é que o identificador clicado se encontra. Para este passo pode ser criado de antemão um *array* que contém o valor inteiro inicial de cada uma das combinações (valor da segunda coluna na tabela 5.9). Depois basta percorrer este *array*, que terá tamanho igual ao número de combinações existentes, e verificar em que índice é que o identificador clicado se encontra. Tendo este índice já sabemos qual é a combinação a que esse mesmo pertence e só precisamos de procurar qual o elemento dentro dessa mesma combinação.

Para a procura dentro de uma combinação, uma solução possível é percorrer todos os grânulos temporais e para cada um deles verificar cada um dos identificadores até encontrar o que se está à procura. Esta solução tem um custo no pior caso de $O(n)$, mas como os identificadores foram gerados de forma sequencial é possível melhorar fazendo a pesquisa binária do elemento, diminuindo assim a complexidade para $O(\log n)$.

Com estas alterações consegui-se criar uma solução que permite deixar de usar as estruturas usadas na API Gisplay. Tem ainda a vantagem de funcionar usando a tecnologia WebGL aproveitando as suas capacidades e de permitir identificar mais de quatro mil milhões de elementos.

Capítulo 6

Avaliação

O capítulo da avaliação estará divido em três secções principais: avaliação demonstrativa, avaliação argumentativa e análise experimental. Na primeira parte serão demonstrados vários cenários que a API Temporal Gisplay permite criar e que são importantes para uma API de mapas temáticos. Na segunda parte temos a avaliação argumentativa onde são apresentados argumentos que demonstram as capacidades da API Temporal Gisplay. Por fim, temos a análise experimental onde é demonstrado o desempenho da API Temporal Gisplay em diferentes contextos e inclusive para conjuntos com milhões de pontos.

6.1 Discussão e avaliação qualitativa

Em termos demonstrativos é de extrema importância referir as principais adições à API Temporal Gisplay durante esta dissertação.

Primeiramente, temos as múltiplas projeções. Através das múltiplas projeções é possível visualizar simultaneamente várias regiões do globo lado a lado. Este é um requisito particularmente importante para um país como Portugal ou os Estados Unidos da América, visto que ambos têm partes do território que fazem parte do país mas que estão separadas do mesmo. Poderá ainda ser usado em contextos em que se pretenda visualizar em simultâneo partes de um país ou zona geográfica, como por exemplo permitir visualizar Portugal Continental, e mais em detalhe, os distritos de Lisboa e do Porto.

Na listagem 6.1 está o código para o *layout* em que o Alaska e as ilhas do Hawaii ficarão do lado esquerdo da visualização e o território continental do Estados Unidos ficará no lado direito. Na figura 6.1a está precisamente demonstrada a visualização que resulta do *layout* apresentado na listagem 6.1. No código do *layout* está explícito que se queria fazer uma divisão vertical em que o lado esquerdo ficará com 40% da largura existente o lado direito os restantes 60%. Depois do lado esquerdo existe nova divisão, agora na

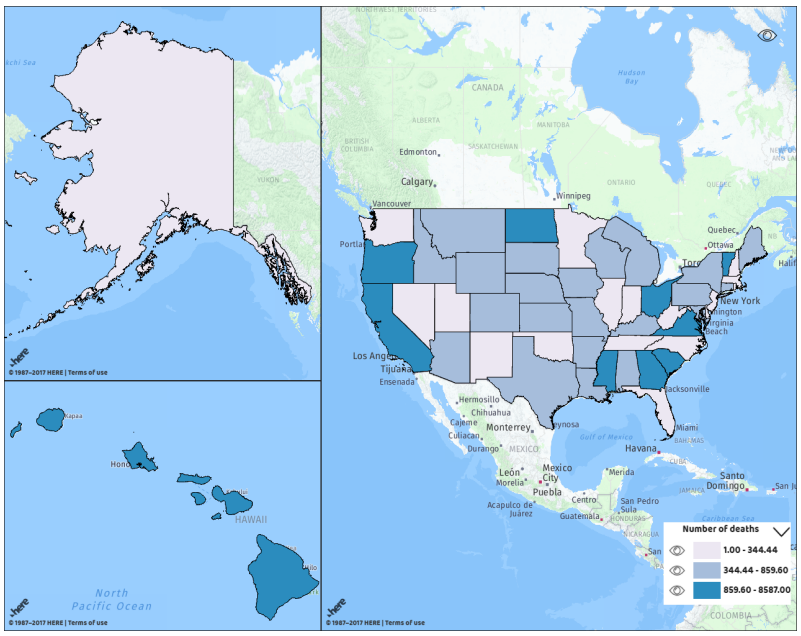
horizontal em que o Alaska vem primeiro e ocupa 60% da altura disponível enquanto que os restantes 40% ficam para o Havaí.

```
bounds: {
    usa: {
        NE: { lng: -67.631836, lat: 48.57479 },
        SW: { lng: -123.793945, lat: 23.563987 }
    },
    alaska: {
        NE: { lng: -138.427734, lat: 72.289067 },
        SW: { lng: -174.638672, lat: 50.176898 }
    },
    hawaii: {
        NE: { lng: -162.388916, lat: 17.056785 },
        SW: { lng: -153.4021, lat: 22.917923 }
    }
},
layout: {
    vertical: {
        sizes: [40, 60],
        descendants: [
            {
                horizontal: {
                    sizes: [60, 40],
                    descendants: ["alaska", "hawaii"]
                }
            },
            "usa"
        ]
    }
}
```

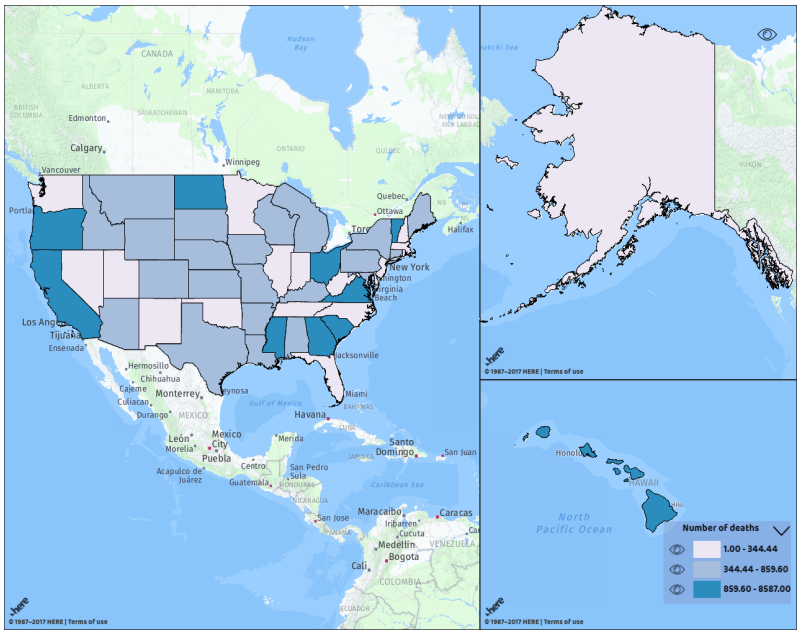
Listagem 6.1: Exemplo do código do *layout* para as múltiplas projeções

```
bounds: {...},
layout: {
     vertical: {
        sizes: [60, 40],
        descendants: [
            "usa",
            {
                horizontal: {
                    sizes: [60, 40],
                    descendants: ["alaska", "hawaii"]
                }
            },
        ]
    }
}
```

Listagem 6.2: Exemplo do código do layout para as múltiplas projeções



(a) Layout da listagem 6.1



(b) Layout da listagem 6.2

Figura 6.1: Exemplo de dois *layout*s diferentes criados com a API Temporal Gisplay.

Na listagem 6.2 é possível ver o código para outro *layout*, sendo que os limites de cada projeção não alteram em relação ao *layout* visto na listagem 6.1. Em relação à listagem 6.1 apenas se altera o código que faz a disposição do *layout*. Com esta pequena alteração de *layout* vai apenas mudar o lado em que cada projeção se encontra. Sendo possível verificar isso mesmo na imagem 6.1b.

Na figura 6.2 é possível ver um exemplo no qual está exposto outro caso com múltiplas projeções, sendo que neste existe uma projeção que engloba todo o território (neste caso

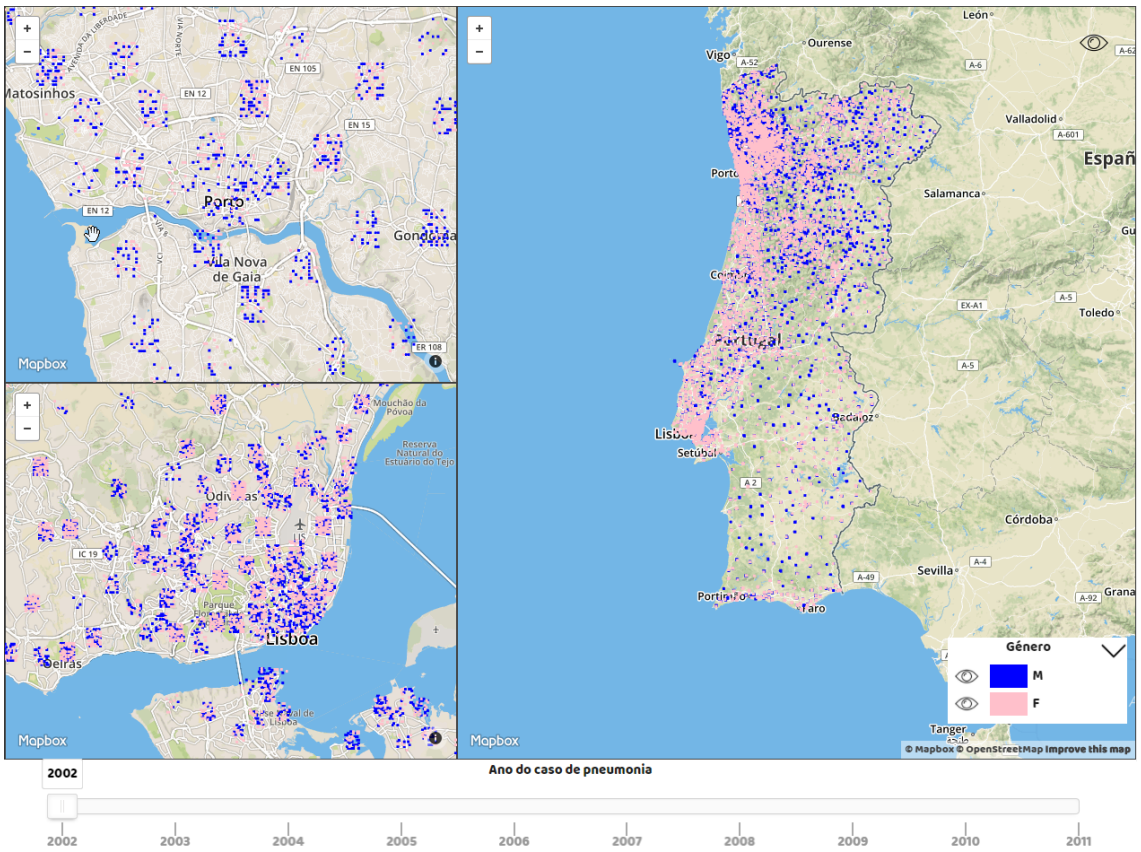


Figura 6.2: Múltiplas projeções com visualização em detalhe de múltiplas zonas geográficas.

Portugal) e duas projeções que permitem ver em mais detalhe o Porto e Lisboa. Em casos como este pode-se querer ver zonas com maior densidade populacional ou que tenham outra propriedade própria que se queira evidenciar.

Outra caraterística muito importante da API Temporal Gisplay é a capacidade de criar visualizações com múltiplas variáveis visuais. Como já foi anteriormente referido, este tipo de visualizações é mais raro do que visualizações com uma única variável visual. Visualizações com variáveis visuais levam à criação de visualizações com mais elementos gráficos e como tal requerem mais capacidade de interpretação por parte do utilizador.

No entanto, há casos em que este tipo de visualização é apropriada, nomeadamente, para conseguir determinar se existe relação entre as variáveis do problema. Na figura 6.3 é possível ver um caso em que existem duas legendas, uma referente a uma variável categórica em que é usada a forma e outra contínua em que é usada a cor. A variável categórica contém informação relativa ao envolvimento ou não de álcool em cada acidente e a contínua tem várias classes que representam o número de mortes resultantes de cada acidente. No caso particular da zona do mapa que estamos a visualizar na figura 6.3, é possível perceber que os acidentes em que resultaram mais mortes não houve sequer álcool envolvido. Isto é possível ver pelo triângulo vermelho e pelo triângulo laranja existentes. Existe ainda a possibilidade de perceber que a maioria dos acidentes existentes,

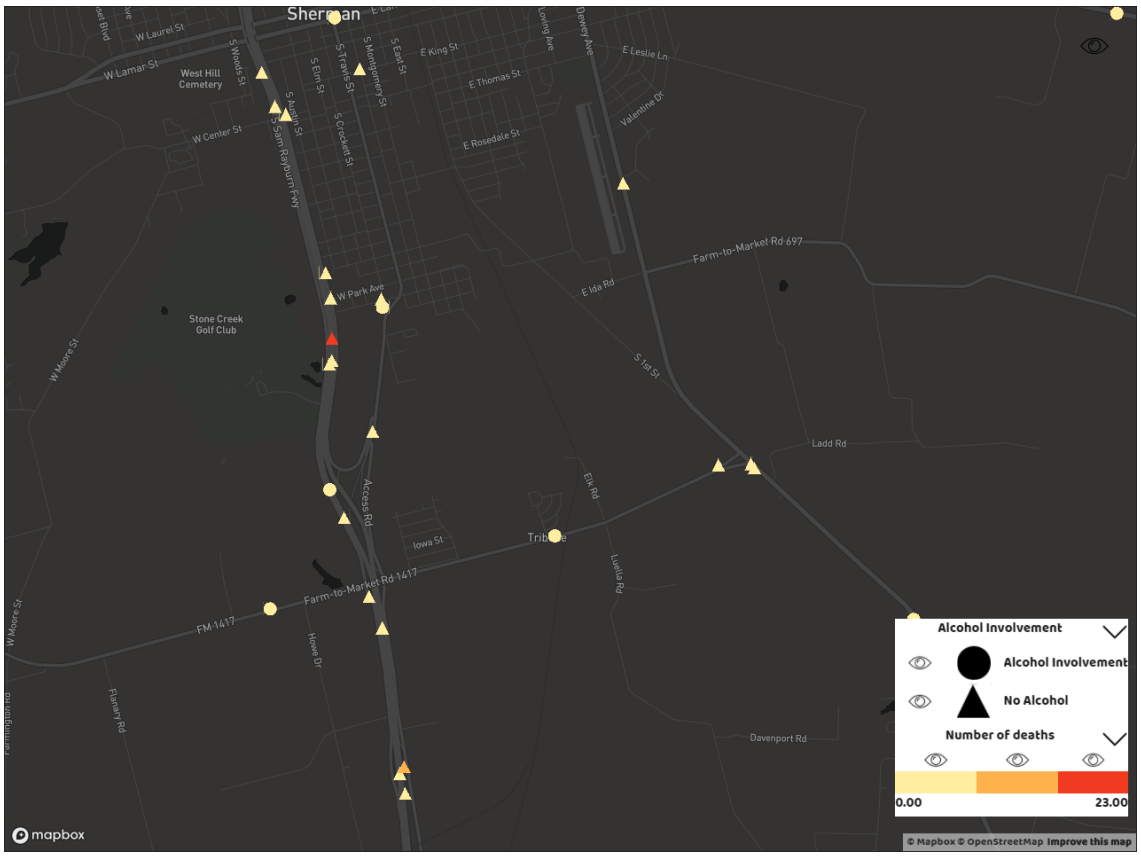


Figura 6.3: Mapa de pontos com múltiplas legendas.

com ou sem envolvimento de álcool, resultam em poucas mortes.

As visualizações com várias variáveis visuais e as múltiplas projeções eram dois objetivos da API Temporal Gisplay. Como é possível verificar pelas figuras 6.1a e 6.1b os *layout*s com múltiplas projeções podem ser facilmente criados e através da figura 6.3 é possível verificar que visualizações com mais do que uma variável visual também não são um problema para a API Temporal Gisplay.

O principal tema desta dissertação era a componente temporal e como é óbvio está presente na API Temporal Gisplay. Na figura 6.2 apresentada anteriormente no contexto de múltiplas projeções tem também presente a componente temporal, na qual é possível verificar que o momento a ser visualizado no mapa é o ano 2002, sendo este um exemplo da visualização de um instante temporal.

Na figura 6.4 é possível ver o mesmo *dataset* do exemplo anterior, mas sendo agora usado o intervalo no controlador temporal. Neste caso em particular está a permitir visualizar todos os casos de pneumonia no intervalo de 2002 a 2011. Isto é possível porque num mapa de pontos tanto se pode utilizar o instante como o intervalo no controlador temporal.

Em termos de uso de cores, a API Temporal Gisplay utiliza nos mapas temáticos disponibilizados o ColorBrewer para fornecer esquemas de cores adequados à visualização.

Os mapas de fundo atualmente disponíveis são o *Google Maps*, *Mapbox*, *Here Maps* e

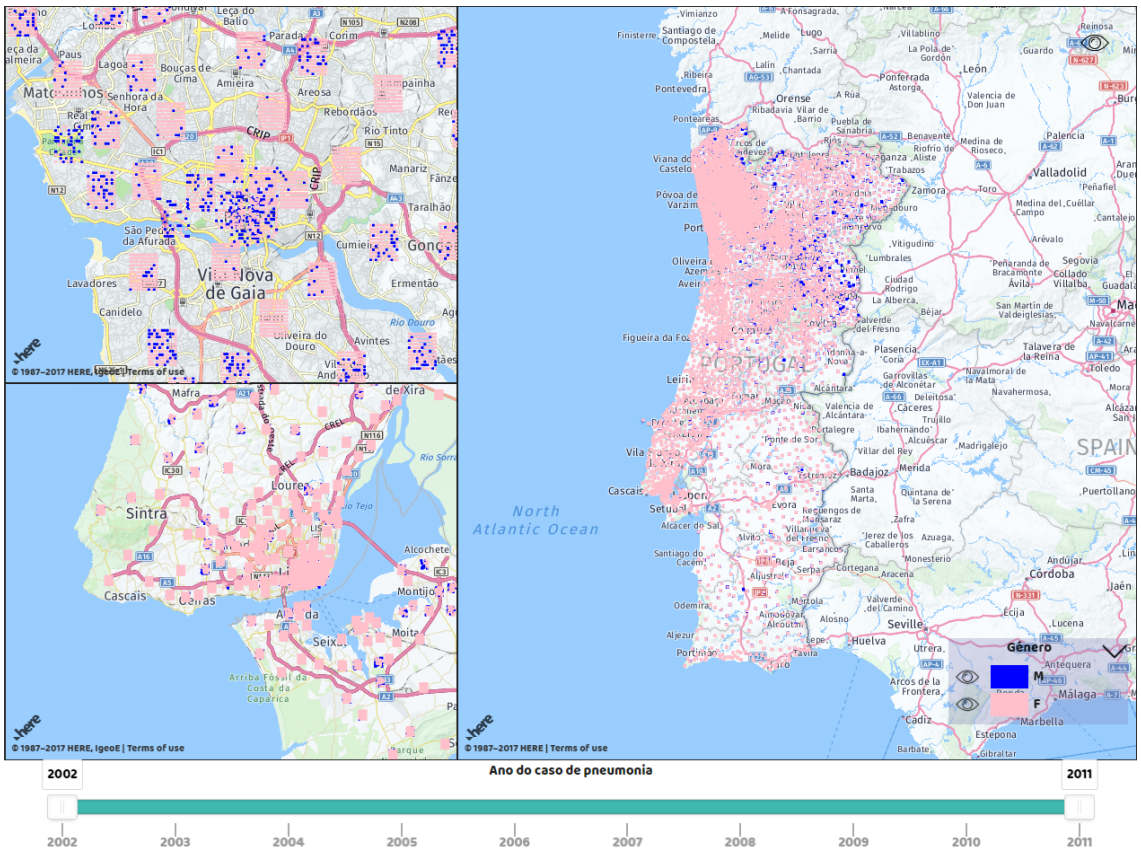


Figura 6.4: Visualização de um intervalo temporal.

Bing Maps. Qualquer um deles pode ser utilizado dando o respetivo nome nas opções globais da API, permitindo assim abstrair a sua criação. Caso existam múltiplas projeções todas elas utilizam o mesmo mapa de fundo e como foi dito anteriormente o programador terá de fornecer os limites de cada uma das projeções.

Mapa temático	draw	drawPicking	defaults	getDefaultColors
Choropleth	7	5	3	1
ChangeMap	-	-	3	1
DotMap	5	5	3	1
ProportionalSymbols	7	7	3	1
FiguresMap	12	12	3	1
LinesMap	2	2	3	1

Tabela 6.1: Linhas de código para os métodos obrigatórios de cada mapa temático fornecido na API Temporal Gisplay.

Como é possível ver na tabela 6.1, os mapas existentes na API Temporal Gisplay requerem pouca quantidade de código para serem implementados. Os métodos necessários são para o desenho das primitivas geométricas respetivas (***draw***), para o *picking* (***drawPicking***), cores e número de classes por omissão (***defaults*** e ***getDefaultColors***), quais os nomes dos ficheiros dos *shaders WebGL* (***getShadersFileNames***), quais as variáveis

visuais suportadas pelo mapa temático (***getAvailableVisualVariables***) e quais os controlos temporais possíveis de utilizar para o mapa temático (***getAvailableTemporalControls***). Os três últimos métodos não estão presentes na tabela porque estes só precisam de uma linha de código, visto que apenas têm de devolver uma lista de elementos baseados em valores existentes na API.

Um dos objetivos para a API Temporal Gisplay era ser facilmente extensível, fornecendo mecanismos simples para criar uma variante de um dos mapas existentes ou produzir um novo de raiz. Como é possível ver o ChangeMap só necessita que se forneça as cores por omissão e o número de classes a utilizar. Isto deve-se ao facto do ChangeMap estar a ser criado com base no Choropleth. Basicamente, o ChangeMap herda tudo o que o Choropleth já possui (incluindo os shaders *WebGL*) e apenas sobrepõe os dois métodos necessários. Assim, podemos verificar que este é outro objetivo que foi cumprido durante esta dissertação. Os outros mapas adicionados precisam de pouco código para a sua implementação visto que usam os métodos já disponibilizados na classe base permitindo assim criar mapas de raiz com relativa facilidade.

Outro objetivo era fornecer um nível alto de abstração, permitindo assim que o uso da API seja conciso e declarativo. Este objetivo foi cumprido porque para usar a API Temporal Gisplay basta fornecer um conjunto de opções (exemplo na secção anterior) e a sua forma de uso é simples e declarativa.

Na próxima secção é demonstrada a capacidade da API Temporal Gisplay em lidar com milhões de pontos e capacidade para fornecer interatividade necessária para facilitar a análise.

6.2 Avaliação Experimental

Todos os resultados nesta análise foram obitdos num desktop com um processador Core i5-4690K 3.5GHz, com 8GB de RAM, placa gráfica AMD R9 290 com 4GB no sistema operativo Windows 10. A API Temporal Gisplay foi construída para ser usada nos browsers mais usados, pelo menos no Chrome, Opera e Firefox, embora os testes tenham sido executados na versão 62.0 do Chrome.

Todos os tempos apresentados estão em milissegundos e são a média de 10 execuções. Todos os tempos foram obtidos através das ferramentas do programador do Chrome. As experimentações foram levadas a cabo usando o mapa de pontos, Choropleth e de linhas, permitindo assim testar a API com todas as primitivas disponíveis. No caso do mapa de pontos e do Choropleth variou-se o tamanho do *dataset* e mediu-se: (*i*) tempo total da API, (*ii*) tempo de desenho do mapa temático, (*iii*) tempo para desenho após operações de interação e (*iv*) tempo de *picking*.

Para o primeiro ponto (tempo total da API) inclui-se todo o tempo que a API leva desde o começo, passando por criar as legendas e controlos temporais até à visualização estar disponível para interação. O segundo ponto é relativo ao tempo do primeiro desenho,

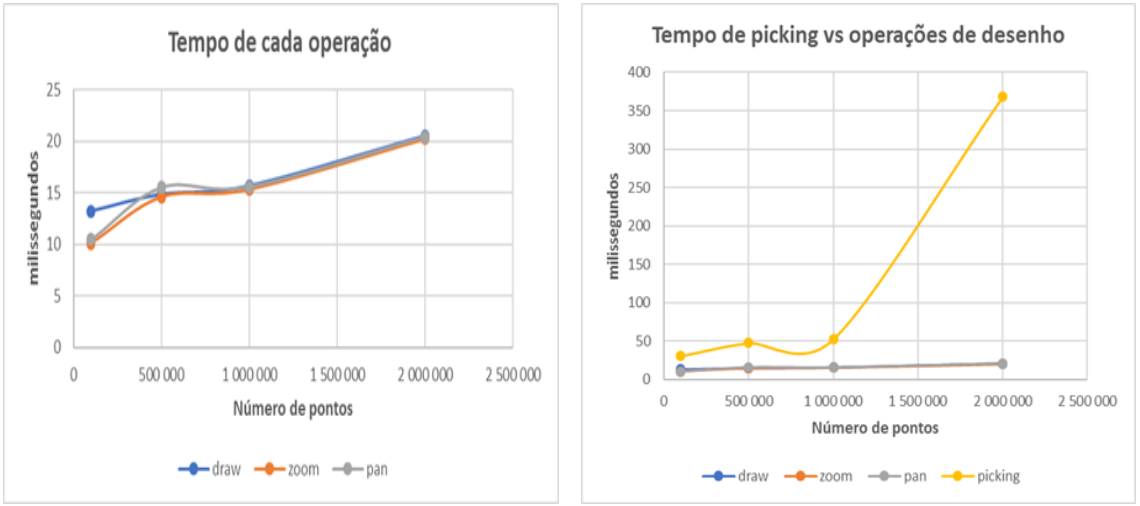
o terceiro ponto refere-se ao tempo de desenho de cada operação de interação e o último ponto é referente ao tempo que a operação de *picking* demora.

Para o mapa de pontos foram testados quatro casos: 100 mil pontos, 500 mil pontos, 1 milhão de pontos e 2 milhões de pontos, permitindo visualizar qual o efeito da variação do volume de dados na performance da API Temporal Gisplay para os quatro pontos acima enunciados.

Figura 6.5: Tempo inicial da API Temporal Gisplay para o mapa de pontos.

Na figura 6.5 é possível ver que o tempo de leitura dos dados cresce de forma linear com o aumento do número de pontos que o *dataset* contém. Isto deve-se principalmente ao Parser CSV que foi criado durante esta dissertação, o qual permite obter uma performance ímpar. No entanto, é possível verificar que o tempo inicial fica próximo de 16 segundos para o caso com 2 milhões de pontos, o que se deve principalmente à quantidade de memória que é necessária quando temos grande quantidade de dados. Esta diferença nota-se particularmente se virmos que para o caso com 1 milhão de pontos a API demora inicialmente aproximadamente 6 segundos enquanto que para 2 milhões de pontos o tempo inicial aproxima-se de 16 segundos.

Na figura 6.6a é possível ver que o tempo de cada operação de desenho aumenta de forma linear com o aumento dos pontos. Por aqui se consegue verificar que a API consegue lidar com grandes volumes de dados, inclusive qualquer operação de desenho para o caso em que existem 2 milhões de pontos demora aproximadamente 20 milissegundos, o que permite sem qualquer problema a interação com o mapa temático.



(a) Tempo de cada operação de desenho.

(b) Tempo de picking vs operações de desenho

Figura 6.6: Tempos de operações de desenho e *picking* para mapas de pontos.

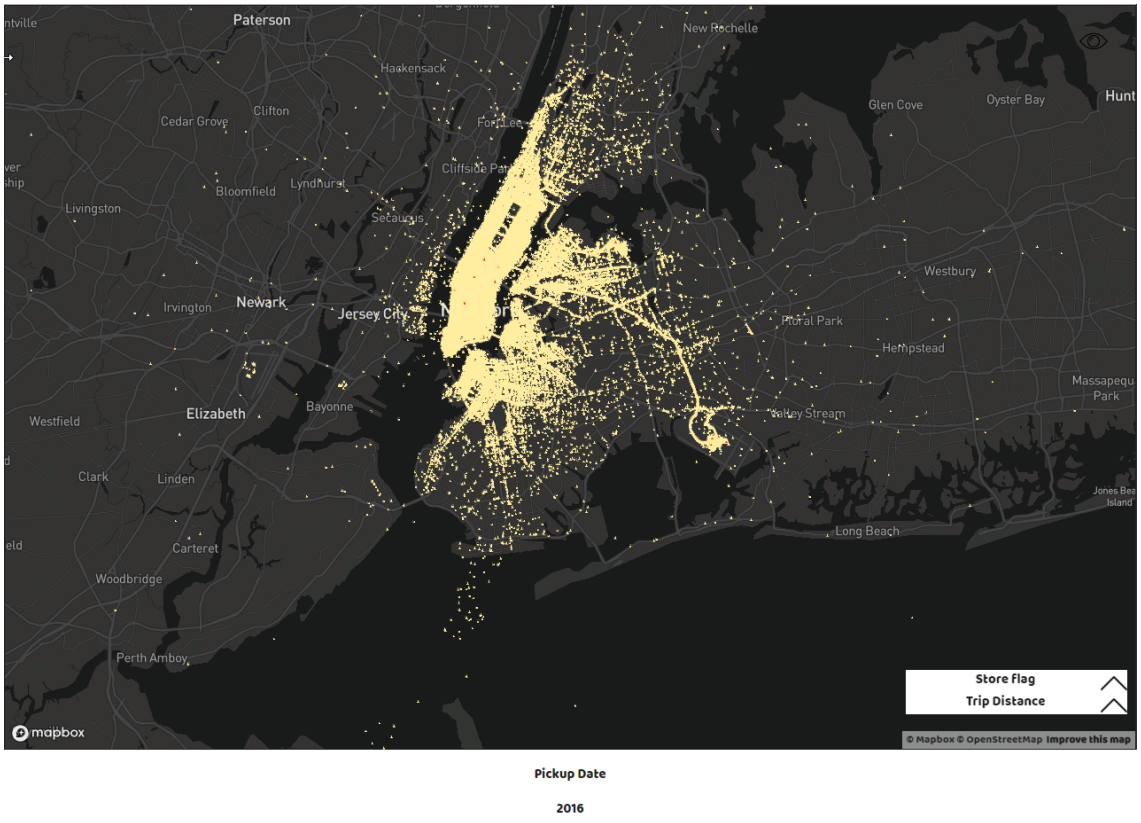


Figura 6.7: Mapa de pontos com 2 milhões de pontos referentes a viagens de táxi em Nova Iorque.

Para os pontos existe ainda a figura 6.6b onde é possível ver os métodos de desenho e o tempo de *picking*, sendo possível fazer comparação entre eles. Como tinha sido referido anteriormente o tempo de desenho é praticamente igual entre as várias operações, mas

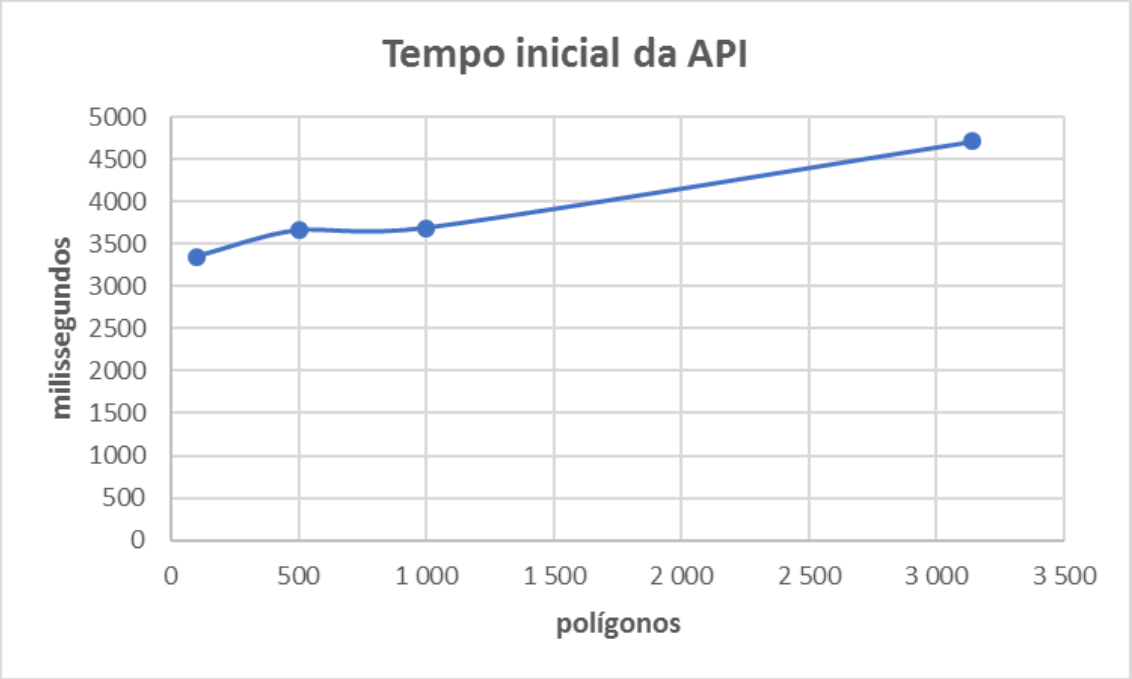


Figura 6.8: Tempo inicial da API Temporal Gisplay para o Choropleth.

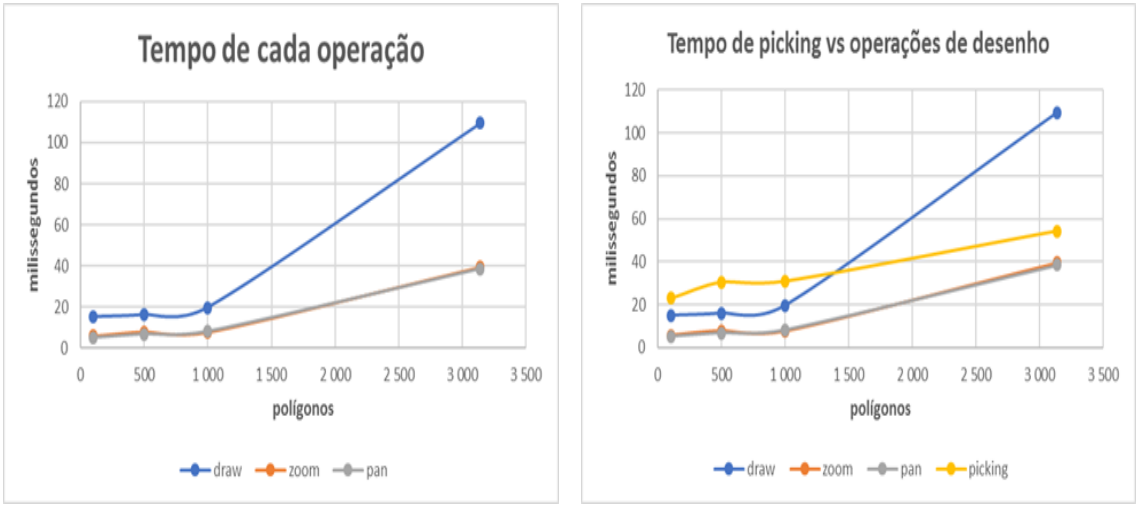
comparados com o tempo da operação de *picking* a diferença é bastante acentuada. O *picking* no caso em que existem 2 milhões de pontos demora acima de 18 vezes mais do que as operações de desenho. Como já foi referido no capítulo de implementação este problema poderá ser resolvido através do uso da pesquisa binária para procura do elemento clicado uma vez que o grosso do tempo de *picking* nos mapas de pontos é o tempo que demora a procurar o elemento após obter o seu identificador. Com esta mudança o tempo do *picking* ficará muito mais próximo do tempo do desenho normal.

Os dados apresentados acima para os pontos resultam do teste de um *dataset*[1] de viagens dos táxis amarelos na cidade de Nova Iorque para o ano de 2016, sendo possível ver o mesmo na figura 6.7. Nos casos testados foi-se variando o número de elementos que o *dataset* continha e foi usado o ano como instante temporal, logo todos os pontos estavam a ser visualizados ao mesmo tempo.

Para o mapa Choropleth foram testados os casos com: 100 polígonos, 500 polígonos, 1000 polígonos e 3141 polígonos.

Na figura 6.8 é possível ver que o tempo de leitura dos dados cresce de forma linear com o aumento do número de polígonos que o *dataset* contém. Para o caso com o número máximo de polígonos testado (3141), este possuía 2 460 570 vértices (pontos), o que é superior aos 2 milhões de pontos que o caso extremo testado para o mapa de pontos, sendo de verificar que no caso do mapa de polígonos o tempo inicial da API é de aproximadamente 4.7 segundos ao passo que no mapa de pontos era de aproximadamente 16

[1] http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml



(a) Tempo de cada operação de desenho.

(b) Tempo de *picking* vs operações de desenho

Figura 6.9: Tempos de operações de desenho e *picking* para o Choropleth.

segundos. Esta diferença deve-se principalmente a que no caso dos pontos existe informação (variáveis visuais, *picking* etc) relativa a cada ponto enquanto que nos polígonos a informação é relativa a cada polígono.

Na figura 6.9a é possível ver que o tempo de cada operação de desenho aumenta de forma constante com o aumento dos polígonos. Para os polígonos existe ainda a figura 6.9b onde é possível ver os métodos de desenho e o tempo de *picking*, sendo possível fazer comparação entre eles. Neste caso o tempo de *picking* é semelhante às outras operações, o que se deve à existência de poucos elementos em relação ao mapa de pontos (no teste do mapa de pontos tínhamos o caso com 2 milhões de elementos enquanto que no mapa de polígonos o caso máximo que temos é de 3141 polígonos). Na figura 6.10 é possível ver o *dataset* que foi usado para os testes feitos para os polígonos, sendo neste caso particular apresentado o caso extremo com 3141 polígonos.

Para o mapa de linhas foi feito um teste com um *dataset* que possuía 56601, sendo estas linhas relativas às principais vias terrestres e marítimas existentes no planeta.

No caso do mapa de linhas apenas foi feito um teste, e neste foi possível aferir que o desenho das linhas ainda não é o mais rápido possível e portanto a performance ainda está aquém do esperado, isto porque atualmente o desenho do mapa de linhas está a ser feito linha a linha. No entanto, é de verificar que num computador que seja de gama média/alta é funcional, e existe interatividade nas operações, o que já não se verifica em dispositivos de gama baixa. Na figura 6.11 é possível ver que o desenho inicial leva, aproximadamente 450 milissegundos, ao passo que as outras operações de interação após esta se situam próximas dos 100 milissegundos. Parte do *dataset* usado para os testes pode ser visto na figura 6.12.

Todos os exemplos testados envolviam apenas um instante temporal. No caso do mapa

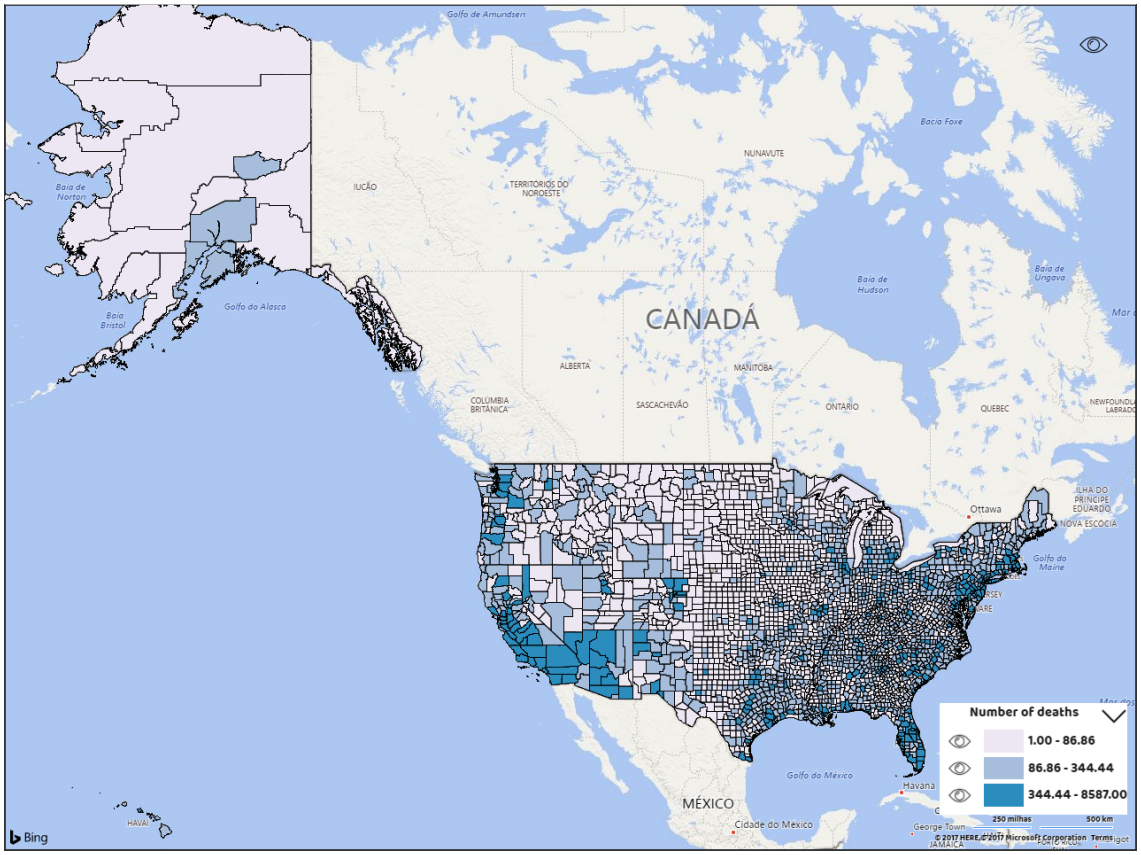


Figura 6.10: Choropleth com 3141 polígonos, sendo cada um deles um condado dos Estados Unidos.

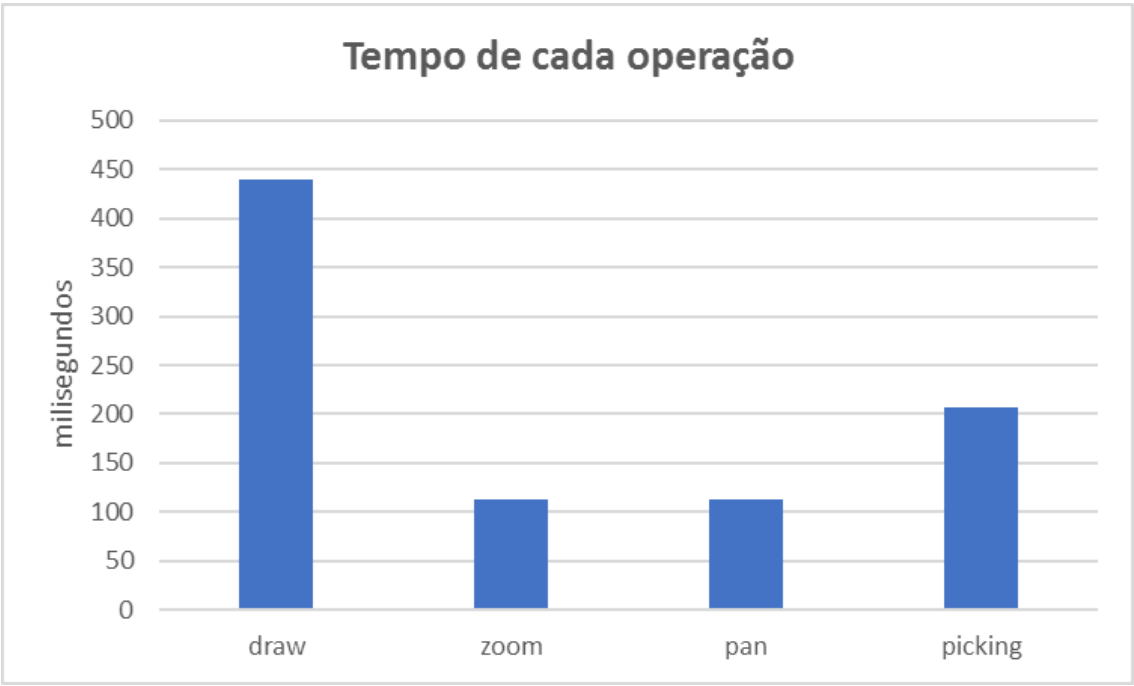


Figura 6.11: Tempo inicial da API Temporal Gisplay para o mapa de linhas.

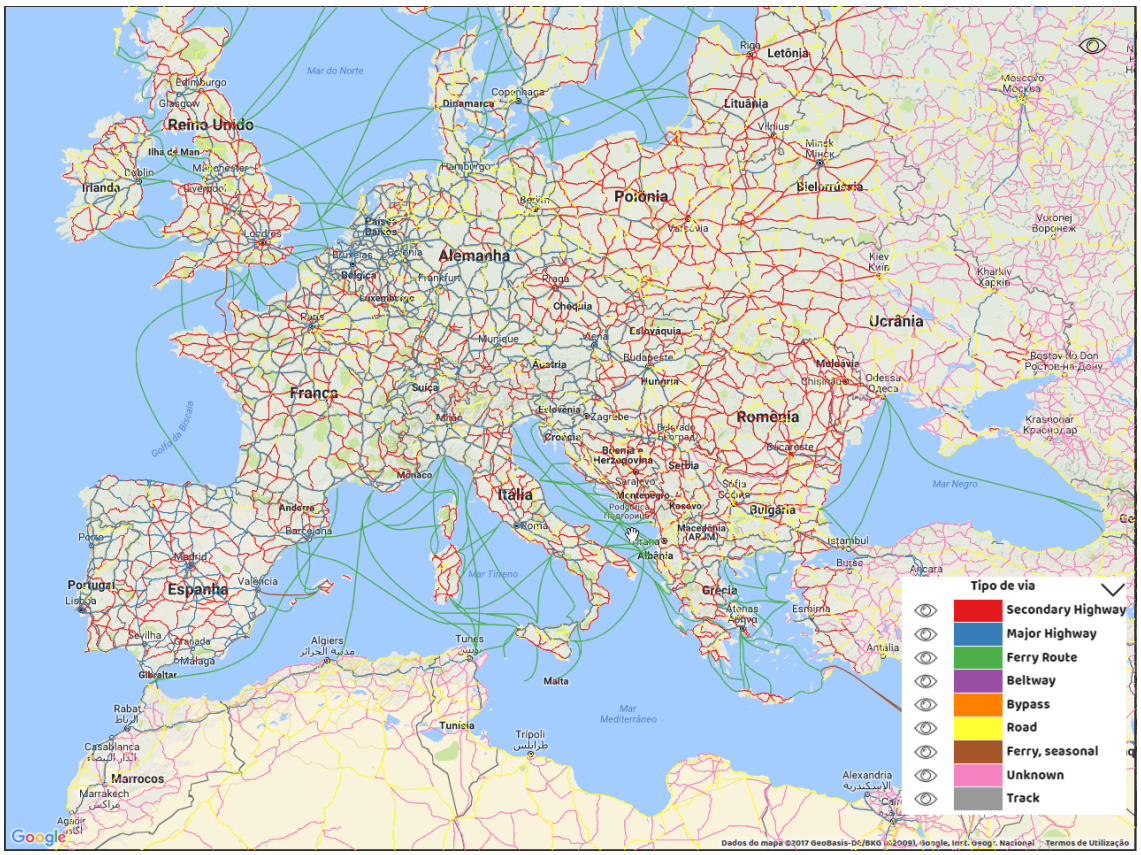


Figura 6.12: Mapa de linhas com as principais vias terrestres e marítimas da Europa e norte de África.

de pontos existiam duas variáveis visuais (cor e forma) o que resultava em seis combinações destas variáveis visuais. No caso do mapa Choropleth apenas existia uma variável visual (cor) com três categorias, enquanto que para o mapa de linhas a cor também é a variável visual usada mas neste caso possui nove categorias diferentes.

Quando existe mais do que um momento temporal, irá existir uma maior divisão dos dados. Sendo assim, se existirem cinco momento temporais então existirá cinco vezes mais separação de dados do que existindo apenas um momento temporal, isto porque os dados estão organizados por combinação de variável visual e dentro de cada combinação pelo respetivo instante temporal. Nos testes efetuados não foi apresentado nenhum exemplo com visualização de um intervalo temporal, no entanto é de referir que o desenho de vários momentos temporais é claramente inferior ao tamanho da janela (número de instantes no intervalo) multiplicado pelo tempo de desenho de um único instante temporal.

6.3 Conclusão

Como podemos aferir pelos resultados acima apresentados a API Temporal Gisplay é capaz de lidar com milhões de pontos e também permite a interação necessária para

facilitar a análise.

Permite o uso de múltiplas variáveis visuais e da criação de uma legenda para cada uma das variáveis do problema. Esta legenda é fácil de utilizar e fornece mecanismos para filtragem.

A API fornece valores por omissão para cada mapa temático (cores e número de classes) mas, no entanto, o programador tem disponível um alto nível de personalização, permitindo sobrepor estes valores.

A API é facilmente extensível e como prova disso existe o ChangeMap que é criado praticamente à custa do Choropleth. Tem um conjunto de mapas que permite ter pelo menos um mapa que utilize cada uma das primitivas existentes (linha, polígono e ponto).

Em termos de mapas de fundo dá a possibilidade de utilizar os mais conhecidos (e.g. Google Maps e Mapbox) dando assim liberdade de escolha ao programador, e acima de tudo, facilitando a adição de novos fornecedores.

Permite visualizar múltiplas projeções sobre os dados, algo que é de grande importância para visualizar regiões afastadas fisicamente no globo.

Sendo o principal motivo desta dissertação, como é óbvio, a API Temporal Gisplay tem a capacidade de utilizar a componente temporal através de um controlador que permite selecionar instantes ou intervalos temporais, fornecendo desta forma a possibilidade de visualizar e analisar fenómenos ao longo do tempo.

Capítulo 7

Conclusões e trabalho futuro

É inequívoco que a API Temporal Gisplay surge na sequência da API Gisplay. Retirando desta API a capacidade de utilizar o WebGL, o facto de ser uma API client-side e de estar pensada em dois níveis, a API de alto nível e a API do nível intermédio. No entanto, para além de endereçar o principal objetivo de incorporar a componente temporal, a API Temporal Gisplay é uma instância completamente nova. A API Temporal Gisplay atinge os seguintes objetivos: utiliza os melhores princípios de visualização de dados permitindo visualização simultânea de múltiplas variáveis visuais, usa os melhores valores por omissão para cálculo de classes e para o uso da cor, é facilmente extensível, tem um alto desempenho permitindo lidar com milhões de pontos em simultâneo e é capaz de lidar com diferentes primitivas, nomeadamente pontos, linhas e polígonos. A API Temporal Gisplay possibilita o uso de múltiplas projeções permitindo visualizar simultaneamente várias zonas geográficas. Fornece ainda mecanismos para o uso de qualquer um dos fornecedores de mapas de fundo mais conhecidos. Atualmente na API Temporal Gisplay são permitidos os mapas: de pontos, símbolos proporcionais, de figuras, Choropleth, ChangeMap e de linhas. Relativamente à componente temporal, permite lidar com dados com entidades ou sem entidades conseguindo poupar memória nos casos em que existe repetição da geografia (e.g. área é usada em vários momento temporais).

Durante esta dissertação foi produzido um artigo[18] no contexto da conferência ICCSA 2017, embora este não contivesse a componente temporal foi feito já com a perspetiva da reimplementação da API, estando previsto submeter novo artigo agora sobre a API Temporal Gisplay.

Para apontar direções de trabalho futuro começo por mencionar um conjunto de melhorias, que apesar de serem de fácil execução, podem ter um grande impacto e que são agora evidentes. Em primeiro lugar, algumas simplificações na API, nomeadamente nas opções que o programador pode fornecer. Depois, temos o caso do *picking*, que como

foi referido pode ser melhorado, principalmente a fase de procura do elemento clicado através do uso de pesquisa binária. A performance do desenho dos polígonos e das linhas poderá ser melhorada fazendo melhor uso das capacidades do *WebGL*, visto que atualmente o desenho está a ser feito elemento a elemento. Melhoria do Parser CSV de modo a aumentar o desempenho através da gestão mais rigorosa da memória permitindo o desenho de mais pontos, sendo isto possível com a passagem da maioria do processamento inicial para os *web workers*.

Naturalmente que no trabalho futuro se podem incluir a introdução de novos mapas na API Temporal Gisplay fazendo uso da extensibilidade permitida pela API. Sugere-se o Heatmap como uma possibilidade pois o mesmo pode vir a fazer uso das capacidades do *WebGL*.

Toda a dissertação assenta na capacidade do *WebGL* para lidar com milhões de pontos. Neste momento está a aparecer a nova versão do *WebGL* (WebGL 2.0) e este poderá ter algumas funcionalidades a explorar pela API Temporal Gisplay. Nomeadamente, *textured arrays* permitindo guardar as formas e os padrões separadamente, em vez de em conjunto, índices para desenho passarem a permitir um máximo de 32 bits em vez dos 16 bits atualmente permitidos e também *uniform buffer objects* que permitem desenho de múltiplas combinações em simultâneo.

A API Temporal Gisplay está disponível em `https://bitbucket.org/staresearch/temporalgisplay`.

Bibliografia

[1] *2010 U.S. Population Dot Density map*. http://robslink.com/SAS/democd59/us_population_density.png. Accessed: 2016-12-26.

[2] V. Agafonkin. *Leaflet*. http://leafletjs.com. Accessed: 2017-02-09.

[3] *Age-standardized prevalence of tobacco smoking among persons aged 15 years and older,2015*. http://gamapserver.who.int/mapLibrary/Files/Maps/Global_Tobacco_use_2015.png. Accessed: 2016-12-26.

[4] W. Aigner e C. Tominski. *Timeviz, Visualization of Time-Oriented Data*. http://timeviz.net/. Accessed: 2017-01-03.

[5] W. Aigner, S. Miksch, H. Schumann e C. Tominski. *Visualization of Time-Oriented Data*. 1st. Springer Publishing Company, Incorporated, 2011. ISBN: 0857290789, 9780857290786.

[6] G. Andrienko. *Flow Map*. https://vcg.informatik.uni-rostock.de/~ct/timeviz/pics/full/flow_map.png. Accessed: 2017-01-03.

[8] N. Andrienko e G. Andrienko. "Interactive Visual Tools to Explore Spatio-temporal Variation". Em: *Proceedings of the Working Conference on Advanced Visual Interfaces*. AVI '04. Gallipoli, Italy: ACM, 2004, pp. 417–420. ISBN: 1-58113-867-9. URL: http://doi.acm.org/10.1145/989863.989940.

[7] N. Andrienko, G. Andrienko e P. Gatalsky. "Journal of Visual Languages Computing Exploratory Spatio-temporal Visualization: an Analytical Review". Em: 2003.

[9] N. V. Andrienko e G. L. Andrienko. "Spatial Generalization and Aggregation of Massive Movement Data". Em: *IEEE Trans. Vis. Comput. Graph.* 17 (2011), pp. 205–219.

[10] P. Bak, F. Mansmann, H. Janetzko e D. A. Keim. "Spatiotemporal Analysis of Sensor Logs using Growth Ring Maps". Em: *IEEE Trans. Vis. Comput. Graph.* 15 (2009), pp. 913–920.

[11] S. E. Battersby, M. P. Finn, E. L. Usery e K. H. Yamamoto. "Implications of Web Mercator and Its Use in Online Mapping". Em: *Cartographica* 49 (2014), pp. 85–101.

[12] J. Bertin e M. Barbut. *Sémiologie graphique: les diagrammes, les réseaux, les cartes*. Gauthier Villars, 1973.

[13] I. Boyandin. *Flowstrates*. https://vcg.informatik.uni-rostock.de/~ct/timeviz/pics/full/flowstrates.png. Accessed: 2017-01-03.

[14] I. Boyandin, E. Bertini, P. Bak e D. Lalanne. "Flowstrates: An Approach for Visual Exploration of Temporal Origin-Destination Data". Em: *Comput. Graph. Forum* 30 (2011), pp. 971–980.

[15] A. Briney. *Thematic Maps in Geography - Overview*. http://geography.about.com/od/understandmaps/a/map-types.htm. Accessed: 2017-01-10.

[16] S. Cabello, H. Haverkort, M. van Kreveld e B. Speckmann. "Algorithmic aspects of proportional symbol maps". Em: *Algorithms–ESA 2006* (2006), pp. 720–731.

[17] D. Cardoso. "Gisplay: A High-Level Client-Side API for Interactive Thematic Maps using WebGL". Tese de mestrado. Portugal: Universidade Nova de Lisboa, 2016.

[18] D. Cardoso, R. Alves, J. M. Pires, F. Birra e R. Silva. "Gisplay- Extensible Web API for Thematic Maps with WebGL". Em: *Computational Science and Its Applications*. Springer International Publishing, 2017. ISBN: 978-3-319-62407-5.

[19] W. Commons. *The world on Mercator projection between 82°S and 82°N. 15° graticule. Imagery is a derivative of NASA's Blue Marble summer month composite with oceans lightened to enhance legibility and contrast. Image created with the Geocart map projection software*. https://upload.wikimedia.org/wikipedia/commons/thumb/f/f4/Mercator_projection_SW.jpg/707px-Mercator_projection_SW.jpg. Accessed: 2017-01-13. 2011.

[20] R. Eccles, T. Kapler, R. Harper e W. Wright. "Stories in GeoTime". Em: *Information Visualization* 7.1 (mar. de 2008), pp. 3–17. ISSN: 1473-8716. URL: http://dx.doi.org/10.1145/1391107.1391109.

[21] *Every person in England and Wales on a map*. https://www.theguardian.com/news/datablog/interactive/2013/aug/01/every-person-in-england-wales-dot-map. Accessed: 2017-02-08.

[22] C. Forlines e K. Wittenburg. "Wakame: sense making of multi-dimensional spatial-temporal data". Em: *Proceedings of the International Conference on Advanced Visual Interfaces*. ACM. 2010, pp. 33–40.

[23] G. Fuchs e H. Schumann. "Visualizing abstract data on maps". Em: *Proceedings. Eighth International Conference on Information Visualisation, 2004. IV 2004.* (2004), pp. 139–144.

[24] E. Gaba. *Deformações na Projeção de Robinson*. https://upload.wikimedia.org/wikipedia/commons/d/dd/Tissot_indicatrix_world_map_Robinson_proj.svg. Accessed: 2017-11-13. 2008.

[25] *Google Maps API*. https://developers.google.com/maps/. Accessed: 2017-02-02.

[26] *Growth Ring Maps*. https://vcg.informatik.uni-rostock.de/~ct/timeviz/pics/full/growth-ring-maps.png. Accessed: 2017-01-03.

[27] D. Guo, J. Chen, A. M. MacEachren e K. Liao. "A Visualization System for Space-Time and Multivariate Patterns (VIS-STAMP)". Em: *IEEE transactions on visualization and computer graphics* 12.6 (2006), pp. 1461–1474.

[28] A. Guttman. "R-trees: A Dynamic Index Structure for Spatial Searching". Em: *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*. SIGMOD '84. Boston, Massachusetts: ACM, 1984, pp. 47–57. ISBN: 0-89791-128-8. DOI: 10.1145/602259.602266. URL: http://doi.acm.org/10.1145/602259.602266.

[29] T. Hägerstraand. *What about people in regional science?* Vol. 24. 1. Wiley Online Library, 1970, pp. 7–24.

[30] M. Harrower e C. A. Brewer. "ColorBrewer. org: an online tool for selecting colour schemes for maps". Em: *The Cartographic Journal* 40.1 (2003), pp. 27–37.

[31] J. Johnson. *Geographic Information*. How to find it, how to use it. Greenwood Press, 2003. ISBN: 9781573563925.

[32] T. Jung. *Deformações na Projeção Azimutal Equidistante*. https://map-projections.net/license/azimutal-equidistant-gpolar:tissot-30-ssw. Accessed: 2017-11-13. 2011.

[33] T. Jung. *Deformações na Projeção de Albers*. https://map-projections.net/license/albers-equal-area-conic:tissot-15-stf. Accessed: 2017-11-13. 2011.

[34] M. Kennedy e S. Kopp. "Understanding map projections". Em: *GIS by ESRI, Environmental System Research Institute* (1994).

[35] T. Kohonen, ed. *Self-organizing Maps*. Springer-Verlag New York, Inc., 1997. ISBN: 3-540-62017-6.

[36] S. Kühn. *Deformações na Projeção de Mercator*. https://commons.wikimedia.org/wiki/File:Tissot_mercator.png. Accessed: 2017-11-15. 2011.

[37] *Leaflet Time Dimension - Add time dimension capabilities on a Leaflet map*. https://github.com/socib/Leaflet.TimeDimension. Accessed: 2017-02-10.

[38] F. Ledermann e G. Gartner. "Mapmap. js: A Data-Driven Web Mapping API for Thematic Cartography". Em: *Proceedings of the 27th International Cartographic Conference (ICC2015)*. 2015.

[39] R. K. Lowe. "Extracting information from an animation during complex visual learning". Em: *European Journal of Psychology of Education* 14.2 (1999), pp. 225–244. ISSN: 1878-5174. URL: http://dx.doi.org/10.1007/BF03172967.

[40] *Map Legend*. http://ohioriverbyway.com/images/FWS_Byway_Legend.2.gif. Accessed: 2017-01-13.

[41] *Map Scale*. https://docs.qgis.org/2.8/pt_BR/_images/map_scale.png. Accessed: 2017-01-13.

[42] *Mapping our changing world - Thematic Maps*. https://www.e-education.psu.edu/geog160/c3_p14.html. Accessed: 2016-12-29.

[43] *Maps of the 2016 US presidential election results*. http://www-personal.umich.edu/~mejn/election/2016/. Accessed: 2017-12-03.

[44] R. McColl. *Encyclopedia of World Geography*. Facts on File Library of World Geography vol. 1. Facts On File, Incorporated, 2014. ISBN: 9780816072293.

[45] A. Mitchell e C. Environmental Systems Research Institute (Redlands. *The ESRI Guide to GIS Analysis: Geographic patterns & relationships*. The ESRI Guide to GIS Analysis. ESRI Press, 1999. ISBN: 9781879102064.

[46] C. M. Morais. *Escalas de Medida, Estatística Descritiva e Inferência Estatística*. http://www.ipb.pt/~cmmm/conteudos/estdescr.pdf. Accessed: 2017-02-04.

[47] *Munsell Colour System*. http://www.paintbasket.com/munsell/munsell_print.jpg. Accessed: 2016-12-26.

[48] M. Newman. *Maps of the 2008 US presidential election results*. http://www-personal.umich.edu/~mejn/election/2008/. Accessed: 2016-12-29.

[49] M. Nöllenburg. "Geographic Visualization". Em: *Human-Centered Visualization Environments Lecture Notes in Computer Science* (), 257–294. DOI: 10.1007/978-3-540-71949-6_6.

[50] G. Pitzl. *Encyclopedia of Human Geography*. ABC-Clio ebook. Greenwood Press, 2004. ISBN: 9780313320101.

[51] R. E. Roth. "Visual Variables". Em: *International Encyclopedia of Geography: People, the Earth, Environment and Technology*. John Wiley Sons, Ltd, 2016. ISBN: 9781118786352. DOI: 10.1002/9781118786352.wbieg0761.

[52] P. Shanbhag, P. Rheingans et al. "Temporal visualization of planning polygons for efficient partitioning of geo-spatial data". Em: *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on*. IEEE. 2005, pp. 211–218.

[53] P. Shanbhag, P. Rheingans e M. desJardins. *Time Oriented Polygons on Maps*. https://vcg.informatik.uni-rostock.de/~ct/timeviz/pics/full/time_polygons.png. Accessed: 2017-01-03.

[54] R. Simmon. *Subtleties of Color - Introduction*. http://earthobservatory.nasa.gov/blogs/elegantfigures/2013/08/05/subtleties-of-color-part-1-of-6/. Accessed: 2016-12-26.

[55] M. Stone. "Choosing colors for data visualization". Em: *Business Intelligence Network* 2 (2006).

[56] D. R. Strebe. *Projeção Azimutal Equidistante*. https://upload.wikimedia.org/wikipedia/commons/e/ec/Azimuthal_equidistant_projection_SW.jpg. Accessed: 2017-11-13. 2011.

[57] D. R. Strebe. *Projeção Cilíndrica de Lamber*. https://commons.wikimedia.org/wiki/File:Lambert_cylindrical_equal-area_projection_SW.jpg. Accessed: 2017-11-15. 2011.

[58] D. R. Strebe. *Projeção Cónica de Lamber*. https://commons.wikimedia.org/wiki/File:Lambert_conformal_conic_projection_SW.jpg. Accessed: 2017-11-15. 2011.

[59] D. R. Strebe. *Projeção de Albers*. https://upload.wikimedia.org/wikipedia/commons/1/1f/Albers_projection_SW.jpg. Accessed: 2017-11-13. 2011.

[60] D. R. Strebe. *Projeção de Lamber*. https://commons.wikimedia.org/wiki/File:Lambert_azimuthal_equal-area_projection_SW.jpg. Accessed: 2017-11-15. 2011.

[61] D. R. Strebe. *Projeção de Robinson*. https://upload.wikimedia.org/wikipedia/commons/9/96/Robinson_projection_SW.jpg. Accessed: 2017-11-13. 2011.

[62] T. Tammet. *SightsMap, a tool that maps the Panoramio photographs taken across the world on a Google Map*. https://www.tnooz.com/wp-content/uploads/2014/01/SightsMap-Tnooz.png. Accessed: 2016-12-29.

[63] S. Thakur e A. J. Hanson. "A 3D Visualization of Multiple Time Series on Maps". Em: *2010 14th International Conference Information Visualisation* (2010), pp. 336–343.

[64] *The Cartographic Process*. https://www.e-education.psu.edu/geog160/node/1882. Accessed: 2016-12-25.

[65] *Thematic Cartography Guide - Bivariate Cartograms*. http://axismaps.github.io/thematic-cartography/articles/bivariate_cartograms.html. Accessed: 2017-01-16.

[66] *Thematic Maps and Visualization of Georeferenced Data*. https://fenix.tecnico.ulisboa.pt/downloadFile/3779576880768/thematic-maps.pdf. Accessed: 2017-01-13.

[67] C. Tominski e H.-J. Schulz. "The Great Wall of Space-Time". Em: *Proceedings of the Workshop on Vision, Modeling Visualization*. The Eurographics Association.

[68] C. Tominski, P. Schulze-Wollgast e H. Schumann. "3D Information Visualization for Time Dependent Data on Maps". Em: *Information Visualisation, 2005. Proceedings. Ninth International Conference on*. IEEE. 2005, pp. 175–181.

[69] C. Tominski, H. Schumann, G. L. Andrienko e N. V. Andrienko. "Stacking-Based Visualization of Trajectory Attribute Data". Em: *IEEE Transactions on visualization and Computer Graphics* 18 (2012), pp. 2565–2574.

[70] *US Oil Consumption - Barrels per year by state*. http://mappingignorance.org/fx/media/2013/12/Figure-11.jpg. Accessed: 2016-12-26.

[71] *Value Flow Map*. https://vcg.informatik.uni-rostock.de/~ct/timeviz/pics/full/value_flow_map.png. Accessed: 2017-01-03.

[72] *VIS-STAMP*. https://vcg.informatik.uni-rostock.de/~ct/timeviz/pics/full/vis-stamp.png. Accessed: 2017-01-03.

[73] *Visual Variables*. http://www.infovis-wiki.net/index.php?title=Visual_Variables#cite_ref-3. Accessed: 2016-12-25.

[74] M. Ward, G. Grinstein e D. Keim. *Interactive Data Visualization: Foundations, Techniques, and Applications, Second Edition*. 360 Degree Business. CRC Press, 2015. ISBN: 9781482257380.